Zig Zag Education

Computer Science

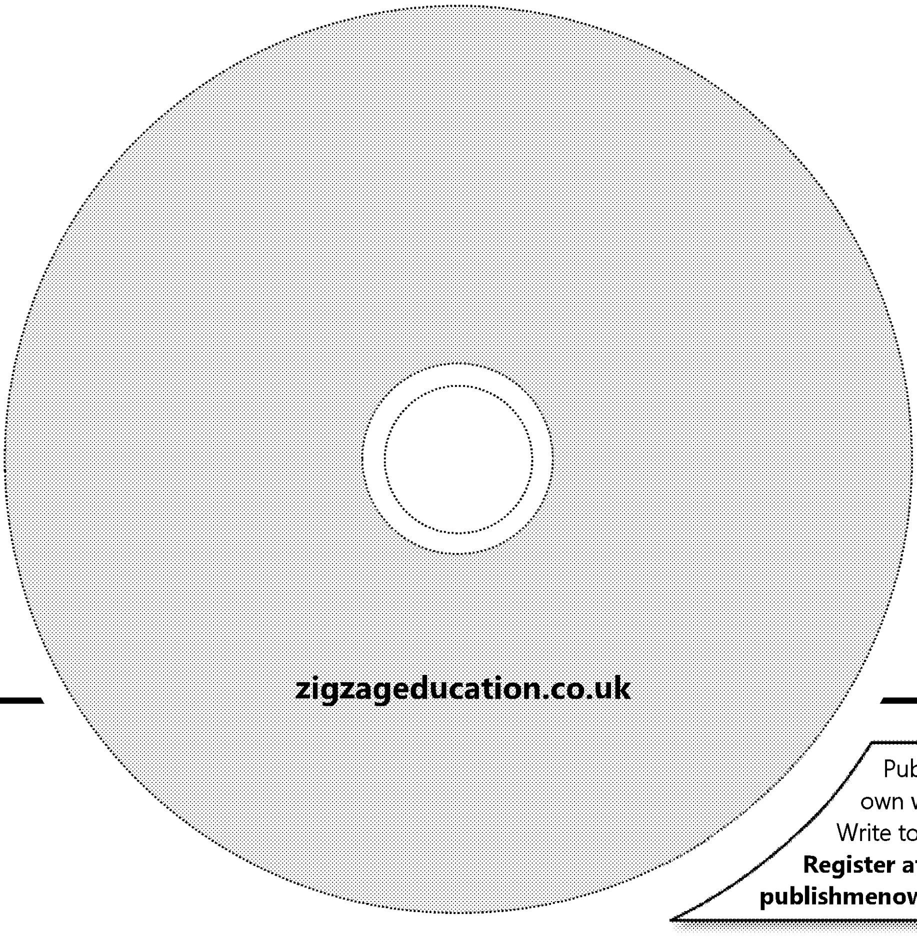
A Level | AQA | 7517

# PAPER 1 EXAM RESOURCE PACK 2018

## WORDS WITH AQA

VB .NET

for A Level AQA Computer Science



CE5/
7895

POD
7895

zigzageducation.co.uk

# Contents

**Printouts of CD resources (for reference)**

# Teacher's Introduction

This resource pack is designed to help you support your students taking the **A Level Computer Science Paper 1** examination. It is based on the **'Words with AQA'** preliminary material (VB .NET) – for examination June 2018.

**New Format:** The biggest improvement in this 2018 resource pack sees all content provided electronically* for the first time. On the CD, you will find the following files.

| | | |
|---|---|---|
| 🗀 | `WordsWithAQA` | for student use – this folder contains all of the content, accessible via a HTML interface |
| 🗀 | `editable` | for teacher use – this folder contains ALL of the documents in editable (docx /pptx) formats |
| 🗎 | `Passwords.txt` | for teacher use – this file contains all of the passwords for the protected PDFs (also listed below) |

* PRINTED COPIES OF ALL THE MATERIALS IN THIS DIGITAL RESOURCE PACK ARE INCLUDED FOR REFERENCE.

**Installation:** Copy the entire `WordsWithAQA` folder onto a network location that is accessible for students, and provide them with a shortcut to the index.html file. All content can be accessed from this page.

**Passwords:** All of the PDFs in the 'Answers & Solutions' HTML page (`answers.html`) are password-protected, so that students can only access them with your permission. Each password is a four-digit code, as follows:

| | | |
|---|---|---|
| 🗎 | `Commentary.pdf` | `1158` |
| 🗎 | `Diagram1Complete.pdf` | `4773` |
| 🗎 | `Diagram2Complete.pdf` | `5382` |
| 🗎 | `Diagram3Complete.pdf` | `3091` |
| 🗎 | `QuestionsMarkScheme.pdf` | `7642` |
| 🗎 | `TaskMarkScheme.pdf` | `2966` |

Should you wish to give students access to ALL protected-PDFs, the master password for all files is:

`zz2ghc4`

The resource pack consists of the following:

① **Pre-release Commentary**, consisting of two parts:

- A general walkthrough of the skeleton program; a written description, flowchart and an animated PowerPoint giving a visual demonstration of the game. It is non-technical in the sense that it doesn't reference or explain any actual code elements – only how the program works when it is run.
- A detailed, technical overview of the skeleton program, describing how all VB .NET subroutines, classes and variables work, including the relationship between them.

**Note:** although this section is intended to give extra support to teachers and students, it should in no way be seen as a substitute to a student exploring the code for themselves. For this reason, this content has been placed on the 'Answers & Solutions' HTML page as a password-protected file, to allow you to control if/when students access it.

② **Structure Diagram Activities**

Three partially complete structure diagram activities for students to complete while getting to grips with the skeleton program. Any missing identifiers, data types, return values, directional arrows, etc. must be added to the diagram. Solutions are provided on the *Answers & Solutions* page as a protected PDF.

③ **Written Questions**

Theory questions testing students' understanding of the 'Words with AQA' code, like Section C in the exam. These questions require access to the skeleton code, but no modifications need to be made to the program. Write-on (with answer lines) and non-write-on version are available format. Solutions are provided on the *Answers & Solutions* page as a protected PDF.

④ **Programming Tasks**

Fifteen modification exercises put students' programming skills to the test, like Section D in the exam. Solutions are provided on the *Answers & Solutions* page as a protected PDF. Note that these are example solutions and you must use your discretion to award marks accordingly where there are valid alternative solutions.

**Free Updates**

Register your email address to receive any future free minor updates made to this resource or other Computing resources your school has purchased, and details of any promotions for your subject.

** resulting from minor specification changes, suggestions from teachers and peer reviews, or occasional errors reported by customers*

**zzed.uk/freeupdates**

An **Electronic Answer Document (EAD)** is provided should you wish students to use it for ③ and/or ④ above.

This resource is intended to supplement your teaching only. **Please read full disclaimer (p. iv) before using it.**

## Introduction

*Words with AQA* is a game in which two human players take turns to make words
have been dealt to them.

When the game begins, a queue of tiles is created, in which 20 tiles are generate
removed from the front of the queue and, once removed, are replaced by an iden
of the queue. The tile queue can be replenished any number of times, so the sam

When the game begins, Player One and Player Two are each assigned 15 tiles tak
their scores are set to 50. An array is also assembled from a text file, which cont
played. The players then take turns, with each turn following this format:

1. The player attempts to play a word using their tiles (each tile can only be p
   if the word 'HAMMER' were to be attempted, the player would need two 'M

   Each letter tile has an integer value, which determines the score, so the wo
   be worth 11 points.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $A_1$ | $B_2$ | $C_2$ | $D_2$ | $E_1$ | $F_3$ | $G_2$ | $H_3$ | $I_1$ |
| $N_1$ | $O_1$ | $P_2$ | $Q_5$ | $R_1$ | $S_1$ | $T_1$ | $U_2$ | $V_3$ |

2. After a word is played, the program checks that it exists in the array of le
   depends on whether the word is allowed.

   a. If the word is allowed, that word's score is calculated using the tile valu
   seven characters long, 5 bonus points are awarded. If the word is eight
   points are awarded. They may then choose how many new tiles they wi
   the following options:

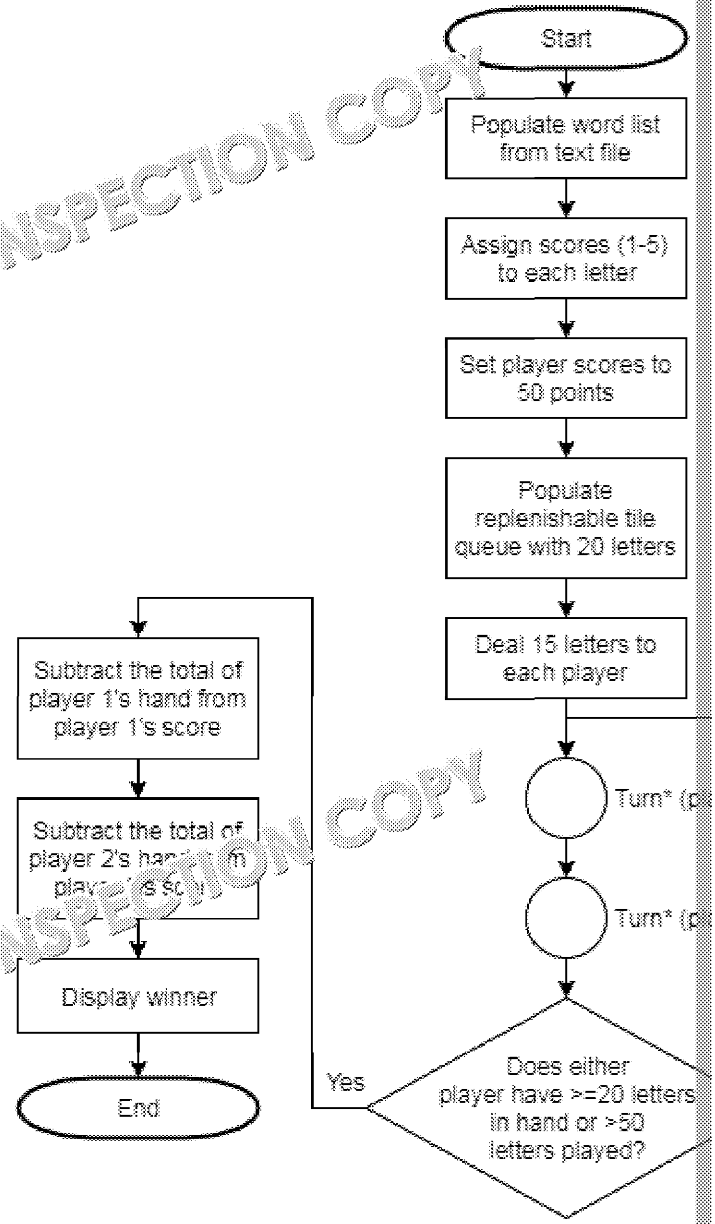
   - three tiles
   - a number of tiles equal to the length of the played word (so four tile
   - a number of tiles equal to the length of the word plus three (so seve
   - no tiles

   b. If the word is not allowed (is not in the array), the player's turn is over, t
   are not permitted to attempt a second word. They are then given three

The game continues until either player has played a total of more than 50 tiles o
more) in their hand. If either of these is true after Player One's turn, Player Two i
game ends.

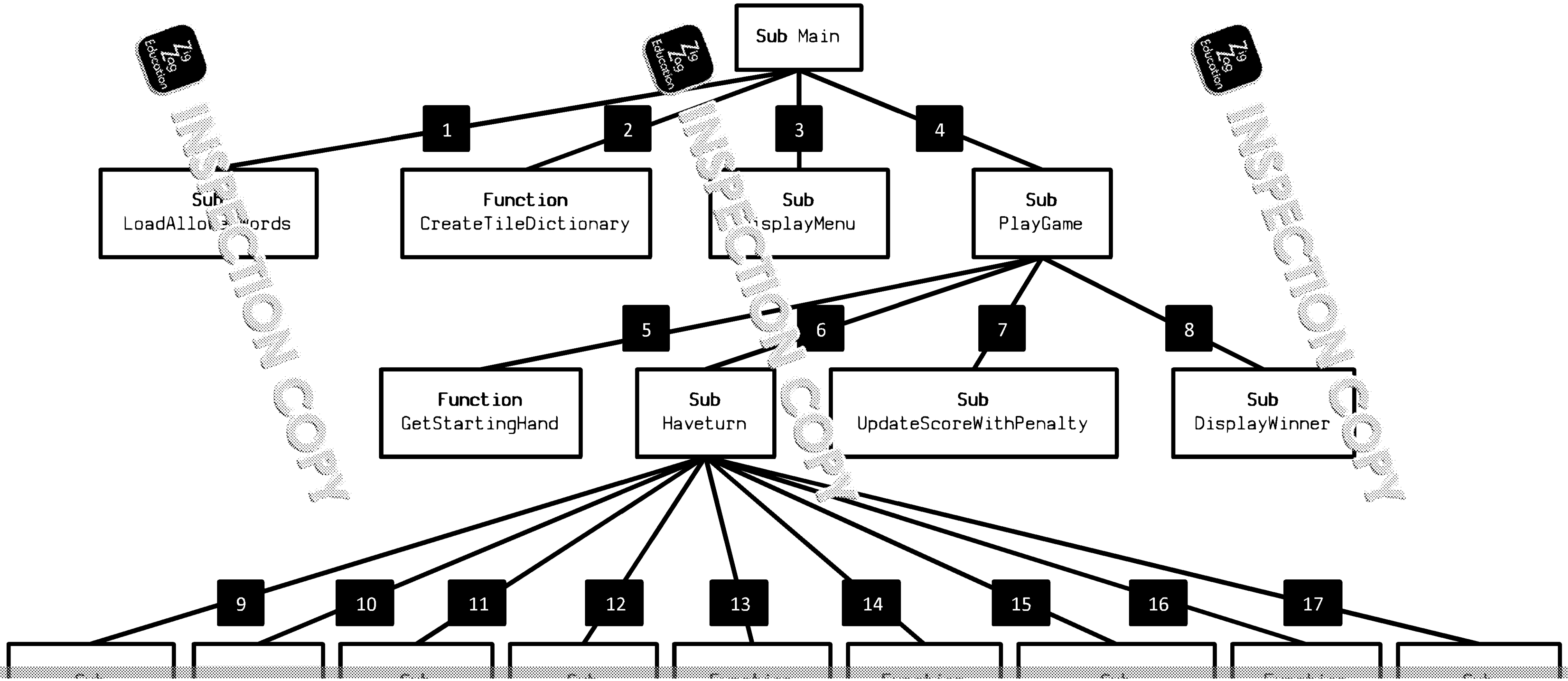
At the end of the game, the total value of each player's hand is subtracted from t
highest score is the winner.

## Program Flowchart


Start
Populate word list from text file
Assign scores (1-5) to each letter
Set player scores to 50 points
Populate replenishable tile queue with 20 letters
Deal 15 letters to each player
Turn* (p
Turn* (p
Does either player have >=20 letters in hand or >50 letters played?
Yes
Subtract the total of player 1's hand from player 1's score
Subtract the total of player 2's hand from player 2's score
Display winner
End

* For the details of the 'turn' subroutines, see steps 1 and 2 on th

## Structure Diagram: Overview



# Subroutine Calls, Parameters and Return Values

The numbers to the left do **not** indicate the order in which subroutines are called, as there are multiple possible orders. Instead, these numbers relate to the numbers in the structure diagram on page 3.

| Call | Parameters | Return |
|---|---|---|
| 1 Main calls LoadAllowedWords | String: AllowedWords | - |
| 2 Main calls CreateTileDictionary | - | Dictionary: TileDictionary |
| 3 Main calls DisplayMenu | - | - |
| 4 Main calls PlayGame | List(String): AllowedWords<br>Dictionary: TileDictionary<br>Boolean: RandomStart<br>Integer: StartHandSize<br>Integer: MaxHandSize<br>Integer: MaxTilesPlayed<br>Integer: NoOfEndOfTurnTiles | - |
| 5 PlayGame calls GetStartingHand | QueueOfTiles: TileQueue<br>Integer: StartHandSize | String: Hand |
| 6 PlayGame calls HaveTurn | String: PlayerName<br>String: PlayerTiles<br>String: PlayerTilesPlayed<br>Integer: PlayerScore<br>Dictionary: tileDictionary<br>QueueOfTiles: TileQueue<br>List(String): AllowedWords | - |

| Call | Parameters | Return |
|---|---|---|
| 11 HaveTurn calls DisplayTileValues | Dictionary: TileDictionary<br>List(String): AllowedWords | - |
| 12 HaveTurn calls FillHandWithTiles | String: PlayerTiles<br>Integer: MaxHandSize<br>QueueOfTiles: TileQueue | - |
| 13 HaveTurn calls CheckWordIsInTiles | String: Word<br>String: PlayerTiles | Boolean: InTiles |
| 14 HaveTurn calls CheckWordIsValid | String: Word<br>List(String): AllowedWords | Boolean: ValidWord |
| 15 HaveTurn calls UpdateAfterAllowedWord | String: PlayerTiles<br>List(String): AllowedWords<br>Integer: PlayerScore<br>Dictionary: TileDictionary<br>String: Word<br>Integer: PlayerTilesPlayed | - |
| 16 HaveTurn calls GetNewTileChoice | - | String: NewTileChoice |
| 17 HaveTurn calls AddEndOfTurnTiles | QueueOfTiles: TileQueue<br>String: PlayerTiles<br>String: NewTileChoice<br>String: Choice | - |
| 18 UpdateAfterAllowedWord calls GetScoreForWord | String: Word<br>Dictionary: tileDictionary | Integer: Score |

## Program Classes

The program contains one module and one class. Their purposes are described briefly in the table below.

| Module/Class | Description |
|---|---|
| `WordsWithAQA` | Most of the code resides in this module, which handles all interaction with the user, validation, determining the flow of the game and identifying game end conditions. |
| `QueueOfTiles` | **Class** to store the structure which contains tiles before they are passed to a player. The structure itself is an array, and an integer points to the rear of the queue. The front is always element zero in the array. |

## Methods

The functions Ⓕ and procedures Ⓟ are described below.

| WordsWithAQA – Methods | Description | |
|---|---|---|
| `AddEndOfTurnTiles` Ⓟ | Parameters: `QueueOfTiles: TileQueue`<br>`String: PlayerTiles`<br>`String: NewTileChoice`<br>`String: Choice`<br>Returns: -<br>CalledFrom: `HaveTurn`<br>Calls: `QueueOfTiles.Add`<br>`QueueOfTiles.Remove` | 1. Declare an integer variable without a value – this will ultimately contain the number of new tiles that will be taken<br>2. If the user has entered option 1 (in ____________), set the integer variable to contain the number of tiles played in the last move<br>3. If the user has entered option 2, set the variable to contain the number 3<br>4. If the user has entered neither 1 nor 2, set the variable to contain the number of tiles played plus 3<br>5. Set up a loop that runs once per tile to be drawn<br>6. Add a character to the string by removing a tile from the front of the |

| WordsWithAQA – Methods | Description | |
|---|---|---|
| `CheckWordIsValid` Ⓕ | Parameters: `String: Word`<br>`List(String): AllowedWords`<br>Returns: `Boolean`<br>Called from: `HaveTurn`<br>Calls: - | 1. Create a Boolean variable set to false<br>2. Loop through each word in the list containing all of the words, stopping only when the end is reached or the attempted word is found<br>3. If a match is found, set the Boolean variable to true<br>4. Return the Boolean to `HaveTurn` |
| `CreateTileDictionary` Ⓕ | Parameters: -<br>Returns: `Dictionary`<br>Called from: `Main`<br>Calls: - | 1. Create an empty map with the format 'character', 'integer'.<br>2. Set up a loop that runs 26 times<br>3. For each iteration, add the corresponding letter to the dictionary (first add A, then add B, etc.)<br>4. For each letter, add the corresponding score (1, 2, 3 or 5), which depends on the letter<br>5. Return the dictionary to `Main` |
| `DisplayMenu` Ⓟ | Parameters: -<br>Returns: -<br>Called from: `Main`<br>Calls: - | 1. Present options to play the game with a random start, training start (string literals) or quit (this subroutine does not accept input or return a value) |
| `DisplayTilesInHand` Ⓟ | Parameters: `String: PlayerTiles`<br>Returns: -<br>Called from: `HaveTurn`<br>Calls: - | 1. Output a blank line<br>2. Output the player's tiles |
| `DisplayTileValues` Ⓟ | Parameters: `Dictionary: TileDictionary`<br>`List(String): AllowedWords`<br>Returns: - | 3. Loop through the `TileDictionary`<br>4. Display each entry in the map in the format 'Points for A: 1', with each entry on a different line |

| WordsWithAQA – Methods | Description | |
|---|---|---|
| FillHandWithTiles Ⓟ | Parameters: QueueOfTiles: TileQueue<br>String: PlayerTiles<br>Integer: maxHandSize<br>Returns: -<br>Called from: HaveTurn<br>Calls: QueueOfTiles.Add<br>QueueOfTiles.Remove | 1. Set up a loop that runs until the size of the player's hand is up to the maximum<br>2. Add a character to the player's hand by removing a tile from the front of the tile queue<br>3. Add a tile to the back of the tile queue |
| GetChoice Ⓕ | Parameters: -<br>Returns: String<br>Called from: HaveTurn<br>Calls: - | 1. Present the player with the mid-game menu<br>2. Prompt the user to select a number or enter a word<br>3. Return this selection to HaveTurn |
| GetNewTileChoice Ⓕ | Parameters: -<br>Returns: String<br>Called from: HaveTurn<br>Calls: - | 1. Declare an empty string<br>2. Prompt the user with a four-option menu<br>3. Return their response to this menu, which will be "1", "2", "3" or "4", to HaveTurn |
| GetScoreForWord Ⓕ | Parameters: String: Word<br>Dictionary: TileDictionary<br>Returns: Integer<br>Called from: UpdateAfterAllowedWord<br>Calls: - | 1. Declare an integer variable and set it to zero<br>2. Loop through each character in the word, adding the score for each letter (taken as read from the dictionary) to the total<br>3. If the length of the word is greater than 7, add 20 to the score<br>4. If the length of the word is 6 or 7, add 5 to the score<br>5. Return the score to UpdateAfterAllowedWord |
| GetStartingHand Ⓕ | Parameters: QueueOfTiles: TileQueue<br>Integer: StartHandSize<br>Returns: String<br>Called from: PlayGame | 1. Create an empty string<br>2. Loop once per tile in StartHandSize (initially 20 times).<br>3. Add a character to the string by removing a tile from the front of the tile queue |

| WordsWithAQA – Methods | Description | |
|---|---|---|
| `HaveTurn` Ⓟ | Parameters: `String: PlayerName`<br>`String: PlayerTiles`<br>`String: PlayerTilesPlayed`<br>`Integer: PlayerScore`<br>`Dictionary: TileDictionary`<br>`QueueOfTiles: TileQueue`<br>`List(String): AllowedWords`<br>`Integer: MaxHandSize`<br>`Integer: NoOfEndOfTurnTiles`<br>Returns: -<br>Called from: `PlayGame`<br>Calls: `DisplayTilesInHand`<br>`GetChoice`<br>`DisplayTileValues`<br>`FillHandWithTiles`<br>`CheckWordIsInTiles`<br>`CheckWordIsValid`<br>`UpdateAfterAllowedWord`<br>`GetNewTileChoice`<br>`AddEndOfTurnTiles` | 1. Display which player's turn it is<br>2. Display's the player's hand<br>3. A loop runs prompting the player to enter option "1", "4", "7", "0" or enter a word, via a call to `GetChoice`, until they enter a word or option "0"<br>4. If they enter "1", the value of all tiles are displayed by calling `DisplayTileValues`<br>5. If they enter "4", the tile queue is displayed by calling `QueueOfTiles.Show`<br>6. If they enter "7", redisplay the player's hand by calling `DisplayTilesInHand`<br>7. If they enter "0", fill the player's hand by calling `FillHandWithTiles`<br>8. If they enter none of those options, the assumption is that they have entered a word, so its length is checked<br>9. If the length is 0, a variable called `ValidWord` is set to false<br>10. If the word is invalid based on a call to `CheckWordIsInTiles`, `ValidWord` is set to false<br>11. If the word is valid, the move is processed by calling `UpdateAfterAllowedWord` and `GetNewTileChoice`<br>12. If the word is invalid, a message is displayed saying 'Not a valid attempt, you lose your turn.'<br>13. New tiles are drawn by calling `AddEndOfTurnTiles` unless the player requested no tiles in the call to `GetNewTileChoice` (if the move was invalid, the player has no choice and three new tiles will be drawn)<br>14. Console outputs the current game state |

| WordsWithAQA – Methods | Description | |
|---|---|---|
| `Main` Ⓟ | Parameters: -<br>Returns: -<br>Called from: -<br>Calls: `LoadAllowedWords`<br>`CreateTileDictionary`<br>`DisplayMenu`<br>`PlayGame` | 1. Game settings are initialised: `MaxHandSize`, `MaxTilesPlayed`, `NoOfEndTurnTiles`, `StartHandSize`<br>2. Menu is displayed and user is prompted<br>3. Loop continues until "9" is entered to quit<br>4. If "1" is entered, game is played with random tiles<br>5. If "2" is entered, game is played with string literals defined in `PlayGame` |
| `PlayGame` Ⓟ | Parameters: `List(String): AllowedWords`<br>`Dictionary: TileDictionary`<br>`Boolean: RandomStart`<br>`Integer: StartHandSize`<br>`Integer: MaxHandSize`<br>`Integer: MaxTilesPlayed`<br>`Integer: NoOfEndOfTurnTiles`<br>Returns: -<br>Called from: `Main`<br>Calls: `GetStartingHand`<br>`HaveTurn`<br>`UpdateScoreWithPenalty`<br>`DisplayWinner`<br>`QueueOfTiles.New` | 1. Set Player One score and Player Two score to 50<br>2. Set number of tiles (for both players) to 0<br>3. Create new tile queue containing 20 tiles<br>4. If a random start has been requested (in `Main`), hands are populated randomly<br>5. Otherwise, hands are populated with string literals<br>6. Loop to run until either player has reached the maximum number of tiles played (50) or the maximum number of tiles in hand (20)<br>7. Call `HaveTurn` alternately for Player One and Player Two until the loop terminates<br>8. Update scores of both players<br>9. Display the winner by calling `DisplayWinner` |
| `UpdateAfterAllowedWord` Ⓟ | Parameters: `String: PlayerTiles`<br>`List(String): AllowedWords`<br>`Integer: PlayerScore`<br>`Dictionary: TileDictionary`<br>`String: Word`<br>`Integer: PlayerTilesPlayed` | 1. Add the length of the word just played to the total number of tiles played<br>2. Loop through each character in the played word, removing a corresponding tile from the player's hand<br>3. Update the player's score by calling `GetScoreForWord` |

| QueueOfTiles – Methods | Description | |
|---|---|---|
| `New` Ⓟ | Parameters: `Integer: MaxSize`<br>Returns: -<br>Called from: `WordsWithAQA.PlayGame`<br>Calls: `Add` | 1. Constructor method – create a new `QueueOfTiles` object when called<br>2. Dimension an array of size `MaxSize` (the parameter)<br>3. Set `Rear` to -1. This variable is the pointer to the back of the queue. Since the array is initially empty, there can be no meaningful rear pointer.<br>4. Call the `add` method repeatedly – e.g. if `MaxSize` is 20, call `add` 20 times. |
| `IsEmpty` Ⓕ | Parameters: -<br>Returns: `Boolean`<br>Called from: `Remove`<br>Calls: - | 1. If `Rear` is -1 (meaning the pointer is not within the array, so the array can be considered empty) return true<br>2. For any other value of `Rear`, return false |
| `Remove` Ⓕ | Parameters: -<br>Returns: `Char`<br>Called from: `WordsWithAQA.GetStartingHand`<br>`WordsWithAQA.AddEndOfTurnTiles`<br>`WordsWithAQA.FillHandWithTiles`<br>Calls: `IsEmpty` | 1. If the list is empty, return a line break character<br>2. Otherwise:<br>a. Store the character at element zero of the array<br>b. Move all other characters in the array back one place, closer to element zero<br>c. Add a line break character to the resulting empty element at the end of the array<br>d. Subtract 1 from `rear`<br>e. Return the character stored in (a) |
| `Add` Ⓟ | Parameters: -<br>Returns: -<br>Called from: `WordsWithAQA.GetStartingHand`<br>`WordsWithAQA.AddEndOfTurnTiles` | *NB This subroutine will do nothing if `rear` already points to the end of the array, since there would be no room to add a character.*<br>1. Generate a random integer from 0 to 25 |

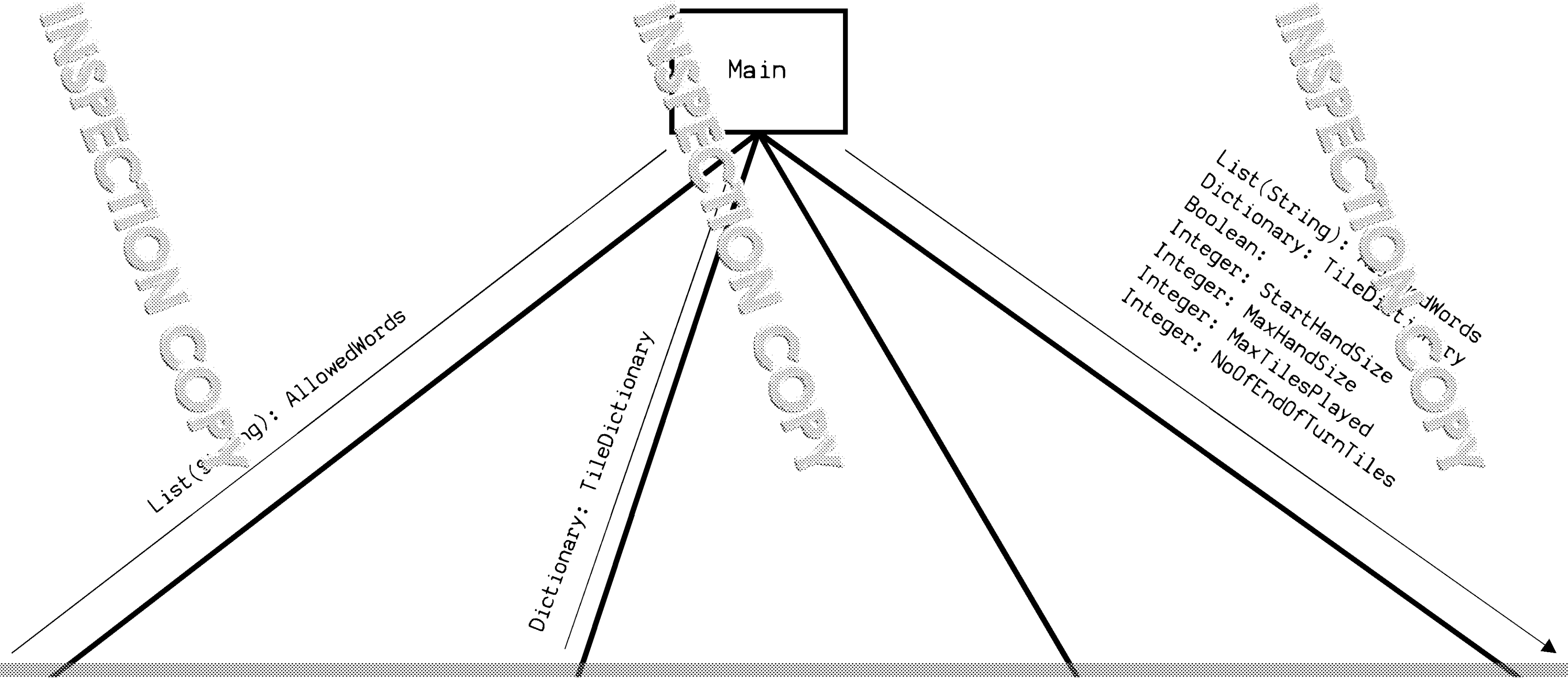
# Variables

The following table contains variables that are declared locally and passed to at least one other method.

| WordsWithAQA Variables | Type | Description | Created in |
|---|---|---|---|
| `AllowedWords` | `List(String)` | A list that contains every valid word that can be played, populated by reading a text file | `Main` |
| `Choice` | `String` | Contains the user input for the main part of their turn, which might be either a word or a menu selection | `HaveTurn` |
| `MaxHandSize` | `Integer` | Any player reaching or exceeding this number of tiles in their hand value ends the game. | `Main` |
| `MaxTilesPlayed` | `Integer` | The threshold for the number of tiles played (by any one player) that ends the game. | `Main` |
| `NewTileChoice` | `String` | Entered via a menu, this indicates the number of tiles the current player would like to take at the end of their turn | `HaveTurn` |
| `NoOfEndOfTurnTiles` | `Integer` | Possibly intended as the default number of new tiles drawn after a turn (set to 3), this variable is passed as a parameter but never actually used. | `Main` |
| `PlayerOneScore` | `Integer` | The number of points, at any point in time, score by Player One | `PlayGame` |
| `PlayerOneTiles` | `String` | A string that stores each tile held by Player One as a character | `PlayGame` |
| `PlayerOneTilesPlayed` | `Integer` | The number of tiles played by Player One | `PlayGame` |
| `PlayerTwoScore` | `Integer` | The number of points, at any point in time, score by Player Two | `PlayGame` |
| `PlayerTwoTiles` | `String` | A string that stores each tile held by Player Two as a character | `PlayGame` |

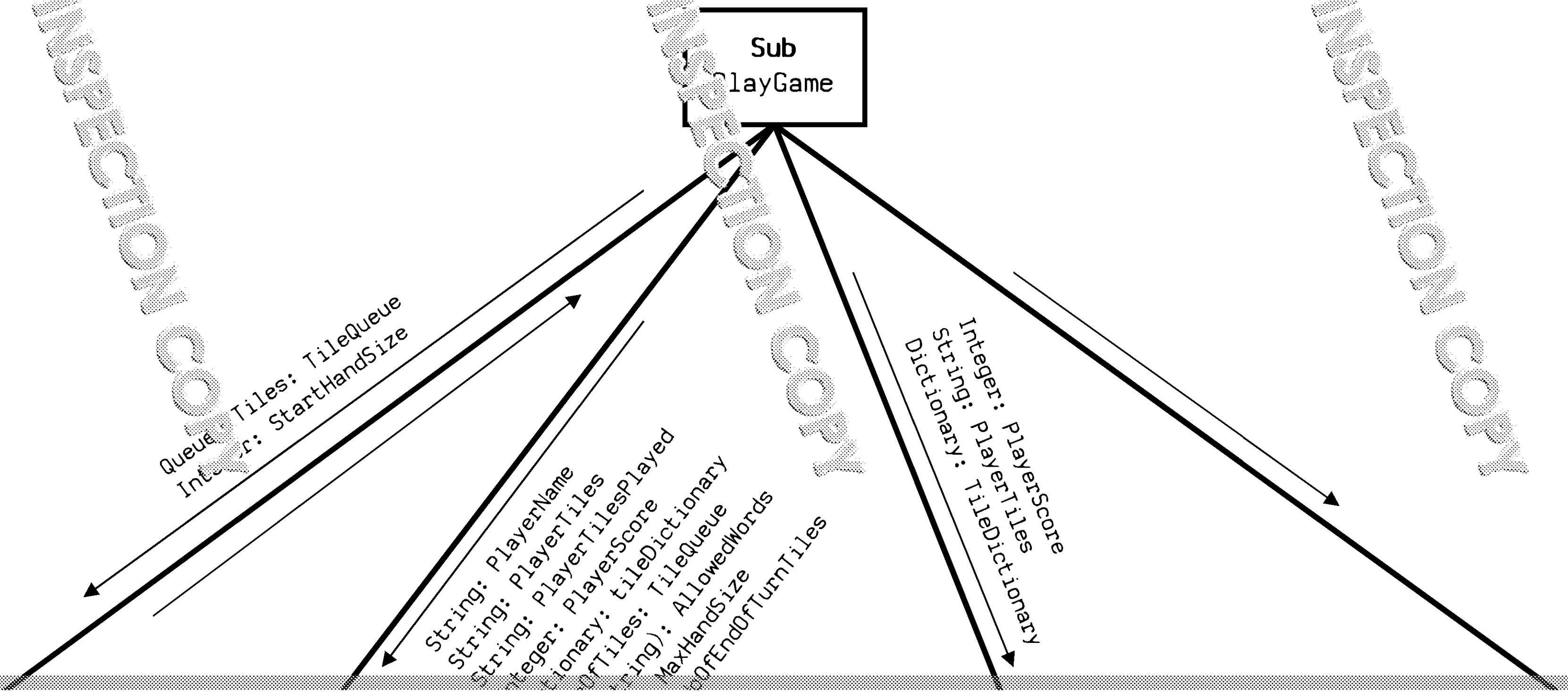
| QueueOfTiles – Variables | Type | Description | Created in |
|---|---|---|---|
| `Contents` | `Char Array` | Character array to contain all letters in the queue before they are passed to a player's hand | N/A (instance variable) |
| `Rear` | `Integer` | The index of the back of the [illegible] used to add new tiles to the correct location in the array. [illegible] a new object of this type is created, the array is empty and this 'Rear' value is -1 | N/A (instance varia[illegible] |
| `MaxSize` | `Integer` | The largest size that the array can [illegible] which is passed to the class's constructor. | N/A (instance variable) |

## **Main** and subroutines called from **Main**


Main
List(String): AllowedWords
Dictionary: TileDictionary
List(String): AllowedWords
Dictionary: TileDictionary
Boolean:
Integer: StartHandSize
Integer: MaxHandSize
Integer: MaxTilesPlayed
Integer: NoOfEndOfTurnTiles

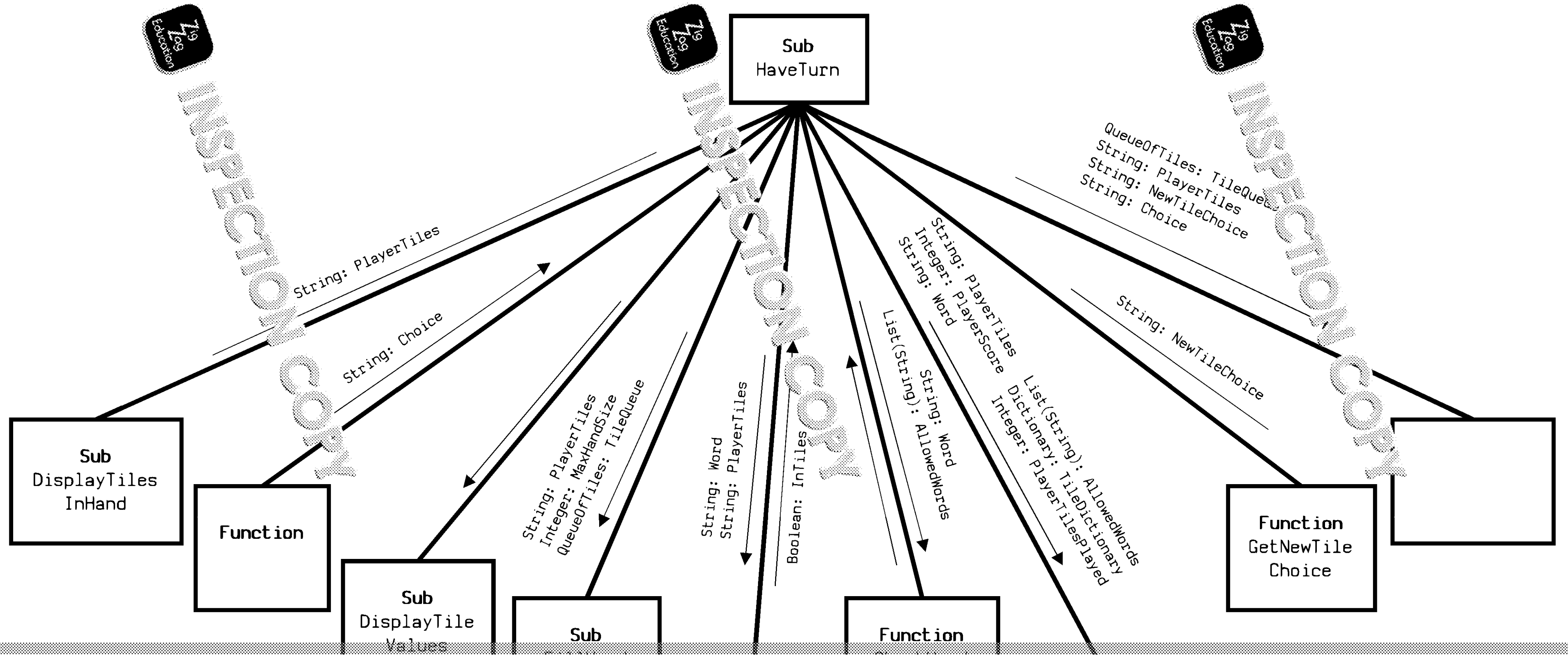
## Structure Diagram Activity 2

## **PlayGame** and subroutines called from **PlayGame**



## Structure Diagram Activity 3

# **HaveTurn** and subroutines called from **HaveTurn**



# Written Questions (VB .NET)

**These questions refer to the preliminary material and require you to load the skel any additional programming**

1. State the name of an identifier for:
    a) A class [1]
    b) An array variable [1]
    c) A variable that is used to store a Boolean return value [1]
    d) A parameter whose data type is dictionary [1]
    e) A function with no parameters [1]
    f) A procedure with no parameters [1]
    g) A local variable within the `HaveTurn` subroutine [1]
    h) An attribute within `QueueOfTiles` [1]
2. Write three lines of code from the skeleton program, each of which calls a di
3. Look at the subroutine `CreateTileDictionary`. Describe the purpose o

```
TileDictionary.Add(Chr(65 + Count),
```

4. State and describe the data structure used by the `CreateTileDictio`
5. Look at the subroutine `CheckWordIsInTiles`. Explain the role of the var
6. Describe the difference between a procedure and a function. State one exam code. [4
7. Describe what would happen if, during a call to `LoadAllowedWords`, the fi found. [3]
8. Describe the actions performed in the following lines of the `LoadAllowedW`

```
AllowedWords.Add(FileReader.ReadLine().Trim().ToUppe
```

9. The `QueueOfTiles` class contains a constructor. Describe what is meant b
10. Describe the effect of the following instruction within the `QueueOfTiles` c

```
Me.MaxSize = MaxSize
```

11. Explain why the variable `Rear` is initialised to -1 in the `QueueOfTiles` co
12. Describe in detail the purpose of the subroutine `UpdateScoreWithPenalt` and/or return values in your answer. [5
13. Describe the operation of the following code within the subroutine `GetScor`

```
Score = 0
For Count = 0 To Len(Word) - 1
    Score += TileDictionary(Word(Count))
Next
```

14. Explain the role of the iterative structure within the subroutine `GetNewTil`
15. Explain why the variable `NewTileChoice` is initialised to the string value "2"

# Written Questions (VB .NET)

**These questions refer to the preliminary material and require you to load the skele any additional programm [illegible]**

1. State the name of an identifier for:

   a) A class [1]

   ..............................................................................................

   b) An array variable [1]

   ..............................................................................................

   c) A variable that is used to store a Boolean return value [1]

   ..............................................................................................

   d) A parameter whose data type is dictionary [1]

   ..............................................................................................

   e) A function with no parameters [1]

   ..............................................................................................

   f) A procedure with no parameters [1]

   ..............................................................................................

   g) A local variable within the `HaveTurn` subroutine [1]

   ..............................................................................................

   h) An attribute within `QueueOfTiles` [1]

   ..............................................................................................

2. Write three lines of code from the skeleton program, each of which calls a di

   ..............................................................................................

   ..............................................................................................

   ..............................................................................................

3. Look at the subroutine `CreateTileDictionary`. Describe the purpose of

```
TileDictionary.Add(Chr(65 + Count), 1)
```

.................................................................................................................................

.................................................................................................................................

.................................................................................................................................

.................................................................................................................................

.................................................................................................................................

4. State and describe the data structure returned by the `CreateTileDictio`

.................................................................................................................................

.................................................................................................................................

.................................................................................................................................

.................................................................................................................................

.................................................................................................................................

5. Look at the subroutine `CheckWordIsInTiles`. Explain the role of the var

.................................................................................................................................

.................................................................................................................................

.................................................................................................................................

.................................................................................................................................

6. Describe the difference between a procedure and a function. State one exam
code. [4]

.................................................................................................................................

.................................................................................................................................

.................................................................................................................................

.................................................................................................................................

.................................................................................................................................

.................................................................................................................................

7. Describe what would happen if, during a call to `LoadAllowedWords`, the fi
found. [3]

.............................................................................................................

.............................................................................................................

.............................................................................................................

.............................................................................................................

.............................................................................................................

8. Describe the actions performed in the following lines of the `LoadAllowedW`

```
AllowedWords.Add(FileReader.ReadLine().Trim().ToUppe
```

.............................................................................................................

.............................................................................................................

.............................................................................................................

.............................................................................................................

.............................................................................................................

9. The `QueueOfTiles` class contains a constructor. Describe what is meant b

.............................................................................................................

.............................................................................................................

.............................................................................................................

10. Describe the effect of the following instruction within the `QueueOfTiles` c

```
Me.MaxSize = MaxSize
```

.............................................................................................................

.............................................................................................................

.............................................................................................................

.............................................................................................................

.............................................................................................................

.............................................................................................................

11. Explain why the variable `Rear` is initialised to -1 in the `QueueOfTiles` con

..........................................................................................................................

..........................................................................................................................

..........................................................................................................................

12. Describe in detail the purpose of the subroutine `UpdateScoreWithPenalt`
and/or return values in your answer. [5

..........................................................................................................................

..........................................................................................................................

..........................................................................................................................

..........................................................................................................................

..........................................................................................................................

..........................................................................................................................

..........................................................................................................................

13. Describe the operation of the following code within the subroutine `GetScor`

```
Score = 0
For Count = 0 To Len(Word) - 1
    Score += TileDictionary(Word(Count))
Next
```

..........................................................................................................................

..........................................................................................................................

..........................................................................................................................

..........................................................................................................................

..........................................................................................................................

14. Explain the role of the iterative structure within the subroutine `GetNewTil`

..........................................................................................................................

..........................................................................................................................

..........................................................................................................................

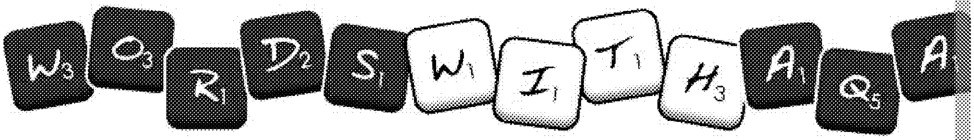
15. Explain why the variable `NewTileChoice` is initialised to the string value “2”

..........................................................................................................................

..........................................................................................................................

..........................................................................................................................

..........................................................................................................................



# Programming Tasks (VB .NET)

**The following questions require you to open the skeleton program and make modi**

## Task 1

This task refers to `GetScoreForWord`.

Currently, the source code assigns bonus points for words that contain more than `GetScoreForWord` so that words of two or three letters incur a one-point pena 'BAR' would normally be worth four points (B=2, A=1, R=1). Following the new ru be applied, meaning 'BAR' would only be worth three points.

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `GetScoreForWord`
- One screen capture showing the word played, the new score and the tota sequence of events:
  - Begin the game with the training hand
  - On *Player One*'s first turn, play the word 'BAL'
  - Press '4' to request no new tiles
- One screen capture showing the word played, the new score and the tota sequence of events:
  - Begin the game again with the training hand
  - On *Player One*'s first turn, play the word 'BARD'
  - Press '4' to request no new tiles

## Task 2

This task refers to `DisplayTilesInHand` and `HaveTurn`.

The program currently displays the letters in the player's hand as a single string. `DisplayTilesInHand` so that each letter is displayed, followed by its points v space; for example:

`A(1) F(3) M(2) E(1)` etc.

Modify `HaveTurn` to allow `DisplayTilesInHand` to have access to the point

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `DisplayTilesInHand`
- Your amended SOURCE CODE PROGRAM for `HaveTurn`
- One screen capture showing *Player One*'s hand at the beginning of the ga hand

## Task 3

This task refers to `CreateTileDictionary`.

Currently, there are no letters worth four points. Modify the code in `CreateTi` letters ‘K’, ‘V’ and ‘Y’ are each worth four points.

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `CreateTileDictionary`
- One screen capture showing the letter values after any player’s turn

## Task 4

This task relates to `PlayGame` and `DisplayWinner`.

‘Words With AQA’ is currently a two-player game. Add code to `PlayGame` to inc should be assigned the same values as *Player One* and *Player Two*. The call to `Di` an additional parameter, and, if *Player Three* has the highest score, they should b

The training hand for *Player Three* should be ‘ABCDEFGHIJKLMNO’.

When displaying the winner, the output should be ‘Player One wins!’, ‘Player Two ‘No clear winner’. This last message should be displayed if any two players are t

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `PlayGame`
- Your amended SOURCE CODE PROGRAM for `DisplayWinner`
- One screen capture showing *Player Three*’s turn and the prompt that marks turn (the action taken by *Player Two* is unimportant). Begin with the trai

## Task 5

This task refers to `HaveTurn`.

Presently, if a valid word is played, the program displays the text ‘valid word’. M followed, on the same line, with the word and its score. If the player has played points, the output should be as follows:

‘Valid word. FARM scores 7 points.’

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `HaveTurn`
- One screen capture showing *Player One*’s first turn with the training hand ABANDONS

## Task 6

This task refers to a new class, `Player`.

The program does not currently allow efficient creation of additional players. Th creation of a class called `Player`.

Create a `Player` class that contains private attributes to store that player's tiles, they have played. There should be a constructor to initialise these attributes to t `PlayGame`. In order to do this, a `QueueOfTiles` object called `tileQueue` wi for the constructor.

Create accessor methods within the class to grant public visibility to each of the a

**Evidence you need to provide:**

- The SOURCE CODE for a new class, `Player`

## Task 7

This task refers to `GetChoice` and `HaveTurn`.

Currently, there is no option for the player to swap their letters. Extend the men option 'Press 2 to swap your letters OR'. Modify `HaveTurn` so that all tiles in th are replaced by an equivalent number of letters from the tile queue. There shou for changing letters.

Once the letters have been swapped, the new hand should be displayed, but pla player.

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `GetChoice`
- Your amended SOURCE CODE PROGRAM for `HaveTurn`
- One screen capture showing *Player One*'s hand both before and after sele 'before' hand should be the training hand.

## Task 8

This task refers to `Add` within the `QueueOfTiles` class.

Modify this subroutine to prevent two consecutive letters being added to the que cause new letters to be generated (and ignored) until a letter is generated that i letter. At that point, the letter should be added to the queue.

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `Add`

## Task 9

This task refers to LoadAllowedWords.

Currently, all words are taken from the aqawords.txt file. Modify the LoadAl
that the following takes place:

1. The user is asked to press option '1' for the default dictionary or '2' for a c
2. If the user presses '1', the aqawords.txt file is used as normal.
3. If the user presses '2', they are prompted for the name of a file. The prog
   the specified file name and this will be used instead of aqawords.txt.

You should only modify LoadAllowedWords in your response to this task. You
validation code and you can assume the user will enter either '1' or '2'.

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for LoadAllowedWords
- One screen capture showing the menu (press '1' for default, press '2' for c
  program's response to '2' being entered.

## Task 10

This task refers to Main.

The program currently attempts to load words from a text file called aqawords.
or not found, the game is allowed to go ahead. Under these circumstances, howe
the game effectively cannot be played.

Modify the Main method so that if LoadAllowedWords results in an empty list
displayed and the program [illegible]

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for Main
- One screen capture showing the entire starting output of the program. P
  should change aqawords.txt in the LoadAllowedWords subroutine

## Task 11

This task refers to `HaveTurn`.

The program currently displays a player's hand, but does not display how many makes the decision of how many tiles to draw more difficult than it needs to be.

Modify `HaveTurn` so that both before and after new tiles are drawn, the number displayed. If no new tiles are drawn, the message should only be displayed once decision not to draw any new tiles. The output should consist of the following f

'You have XX tiles remaining.'

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `HaveTurn`
- One screen capture showing the output having done the following:
  - Selected '2' to play with the training hand
  - Played the word BAT for *Player One*
  - Pressed '3' to indicate that you want to replace the tiles played
- One screen capture showing the output having done the following:
  - Selected '2' to play with the training hand
  - Played the word BRAT for *Player One*
  - Pressed '4' to indicate that you want no replacement tiles

## Task 12

This task refers to a new class, `LetterTile`.

The program currently uses characters to represent tiles, with the value of each structure. An alternative would be to create a new class, from which objects cou representing a tile.

Create a new class called `LetterTile`. It should be assembled according to th

- There should be three private attributes – `letter` (char), `score` (int) a attribute would be set to 'true' for a vowel (A, E, I, O, U) and 'false' for a c
- The constructor should have two parameters – the letter and the diction letter. The constructor should set all three attributes correctly. For exam attributes would be set as follows:
  - `letter`: A (the letter should always be stored in the attribute in
  - `score`: 1 (since 'A' is worth a single point)
  - `isVowel`: true (since 'A' is a vowel)
- There should be a public accessor method to grant access to each attrib

**Evidence you need to provide:**

- The SOURCE CODE for a new class `LetterTile`.

## Task 13

This task refers to `Main` and `DisplayMenu`.

Presently, the values of `MaxHandSize`, `MaxTilesPlayed`, `NoOfEndOfTurnT` written into the code and cannot be changed by the user.

- Add a menu option in `DisplayMenu` to read '3. Settings'.
- Alter the code in `Main` so that if '3' is chosen from the menu, the user wi following prompts in turn:
    - Enter maximum hand size:
    - Enter maximum tiles played:
    - Enter default new tiles:
    - Enter starting hand size:
- The user input (which will not need to be validated) should go into the v `MaxTilesPlayed`, `NoOfEndOfTurnTiles` and `StartHandSize` res

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `DisplayMenu`
- Your amended SOURCE CODE PROGRAM for `Main`
- One screen capture showing the following:
    - Option 3 is chosen from the first menu
    - Set maximum hand size to 25
    - Set maximum tiles played to 40
    - Set default new tiles to 2
    - Set starting hand size to 10
    - Begin the game with a random start hand

## Task 14

This task refers to the `QueueOfTiles` class.

Presently, letters are chosen purely at random, which can result in a queue and a sufficient number of vowels.

Modify the add subroutine and any other necessary code so that letters are selec order:

- The first letter is chosen purely at random, as is currently the case
- The second letter is also chosen at random
- The third letter is chosen at random from the vowels only (A, E, I, O, U)
- After this, every third letter should be a randomly chosen vowel

It may help you to know that the ASCII values for the vowels are as follows: A=65

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for the `QueueOfTiles` class
- One screenshot showing *Player One*'s tiles after choosing to play with a ra

## Task 15a

This task refers to `HaveTurn` and a new subroutine called `ResolveBlanks`.

The game is currently played without blank tiles. In other games, such as Scrabb part of a word. That blank tile can count for any letter as long as it results in the

Create a new subroutine called `ResolveBlanks`. This subroutine should behav

- It accepts a single variable word, which is a string that may or may not c play the role of blanks
- For each dash encountered, the user is given the prompt 'Enter value of b identical for all instances of dash in the word)
- The user inputs a character, which replaces a dash (in the event of multip should be replaced in the order in which they appear). No validation is re
- The word, now without any dashes, is returned to `HaveTurn`

You should also modify `HaveTurn` so that `ResolveBlanks` is called immediate `CheckWordIsValid` is called.

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `HaveTurn`
- Your SOURCE CODE PROGRAM for the new `ResolveBlanks` subroutine
- One screenshot showing the output that results from the following:
    - Before running the program, change the following line in the `Pla`
        - FROM: `PlayerOneTiles = "BTAHANDENONSARJ"`
        - TO: `PlayerOneTiles = "--HANDENONSARJ"`

        (i.e. change the first two characters to dashes)
    - Run the program and select the training hand option
    - Play the following: [illegible]
- One screenshot showing the output that results from the following:
    - Before running the program, ensure that the line of code has bee
    - Run the program and select the training hand option
    - Play the following: han--
    - At the first prompt, enter `D`
    - At the second prompt, enter `Y`

## Task 15b

This task refers to `HaveTurn` and `UpdateAfterAllowedWord`, and assumes t
this task is working correctly.

Currently, when a blank tile is played, it is not removed from the player's hand. I
HANDY tries to play the word 'handy' using the blanks, the letter tiles 'H', 'A', 'N', '
player's hand, and the player keeps the blanks.

Modify the program so that the score is still determined based on the letters tha
any blank tiles that are used are now removed from the player's hand.

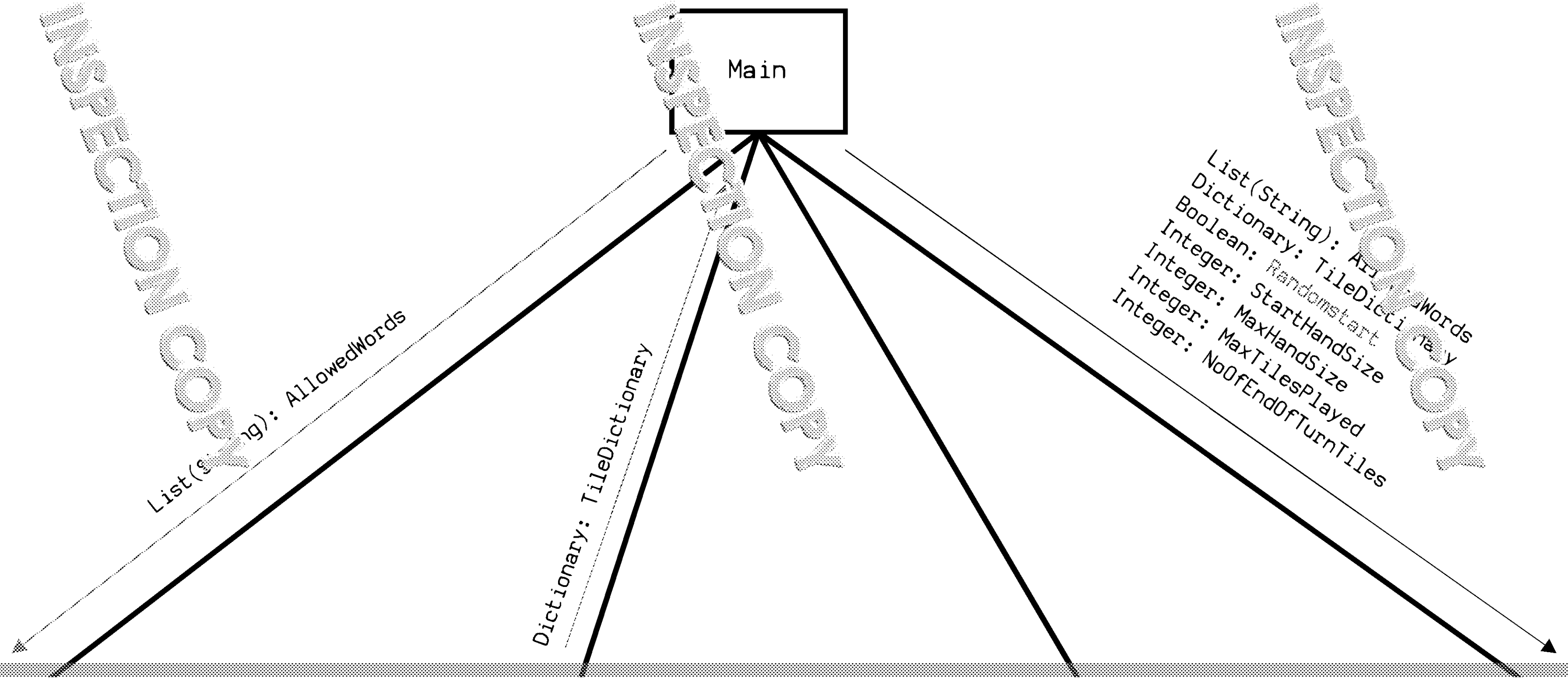
Before running the program, ensure that the value of `PlayerOneTiles` is still s

**Evidence you need to provide:**

- Your amended SOURCE CODE PROGRAM for `HaveTurn`
- Your amended SOURCE CODE PROGRAM for `UpdateAfterAllowedWo`
- One screenshot showing the output that results from the following:
    - Run the program and select the training hand option
    - Play the following: han--
    - At the first prompt, enter `D`
    - At the second prompt, enter `Y`
    - Select option 4 to take no new tiles
    - Continue running the game, entering any inputs, until *Player One'* turn is shown.

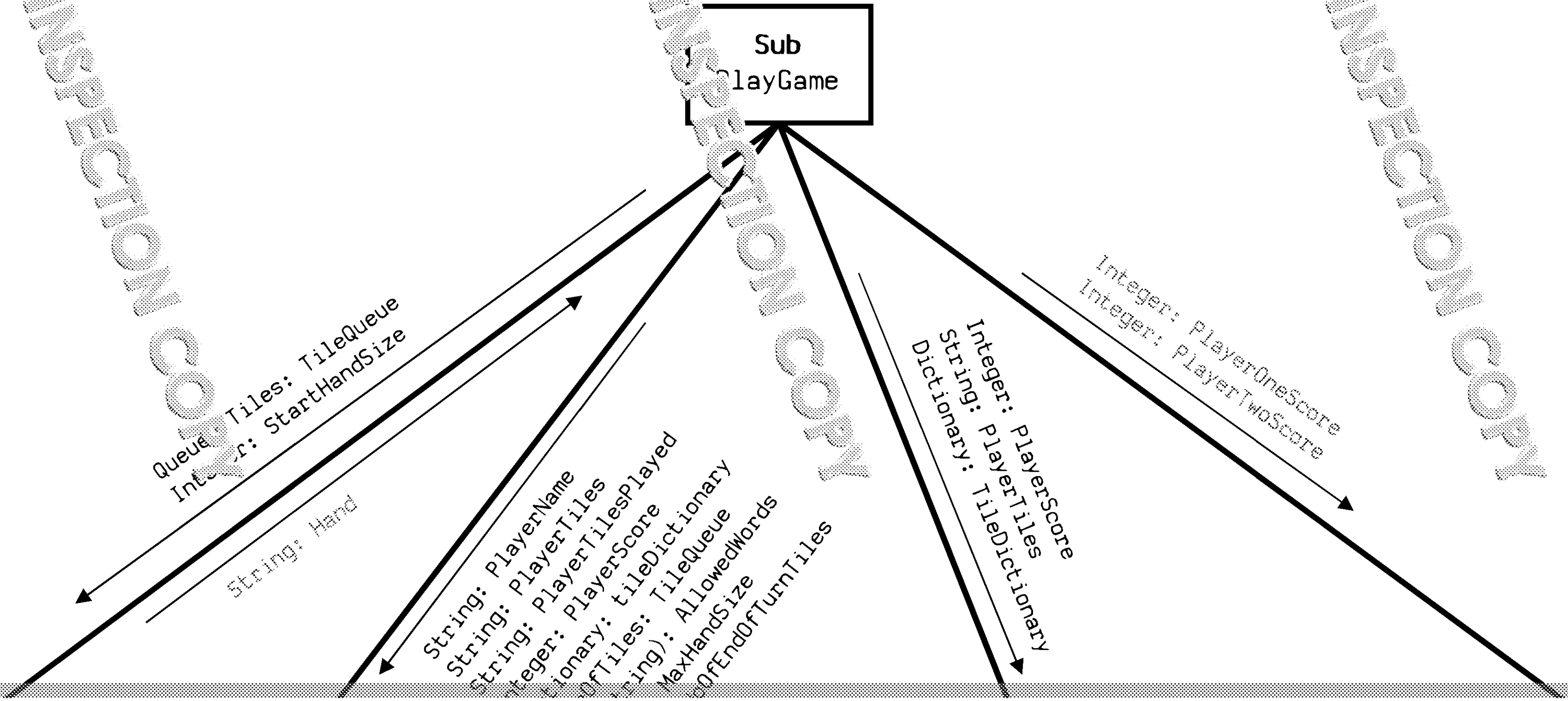
**Structure Diagram 1**

## **Main** and subroutines called from **Main**


Main
List(String): AllowedWords
Dictionary: TileDictionary
List(String): AllowedWords
Dictionary: TileDictionary
Boolean: RandomStart
Integer: StartHandSize
Integer: MaxHandSize
Integer: MaxTilesPlayed
Integer: NoOfEndOfTurnTiles

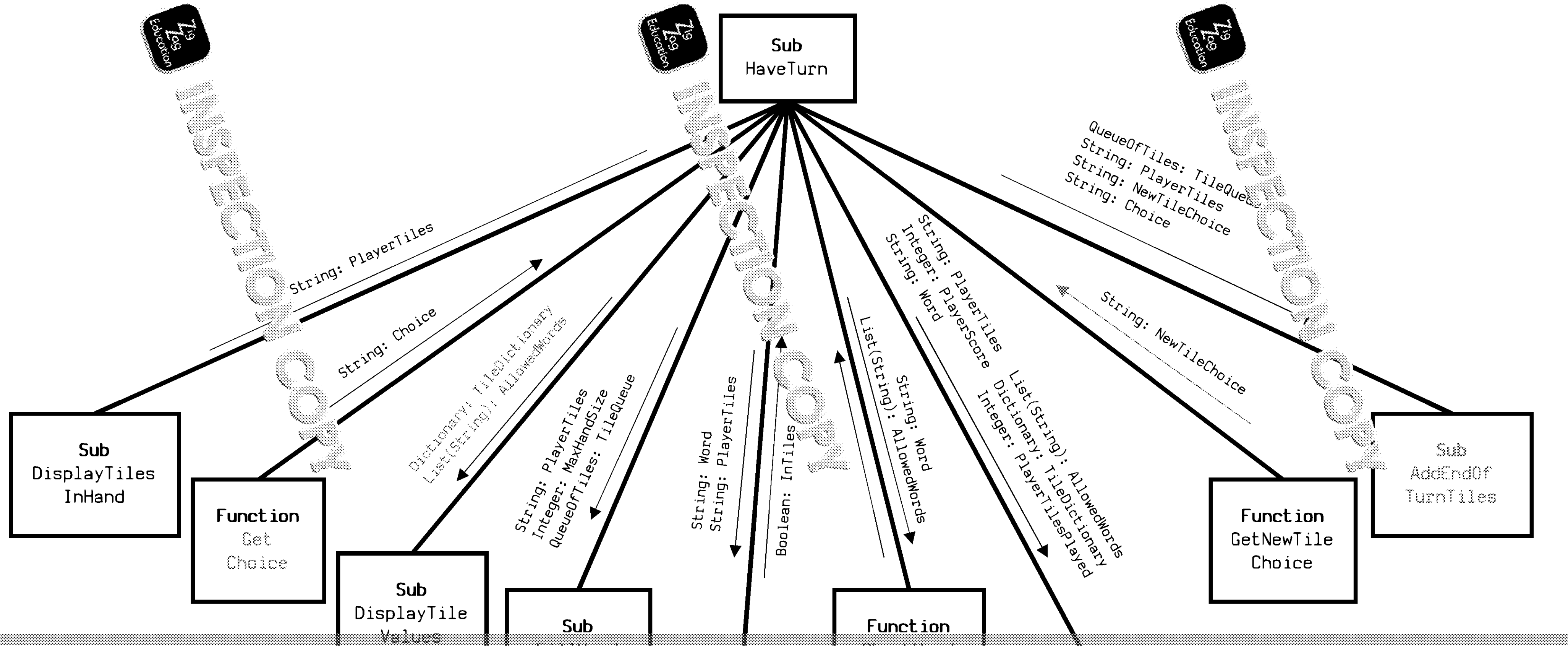
## Structure Diagram 2

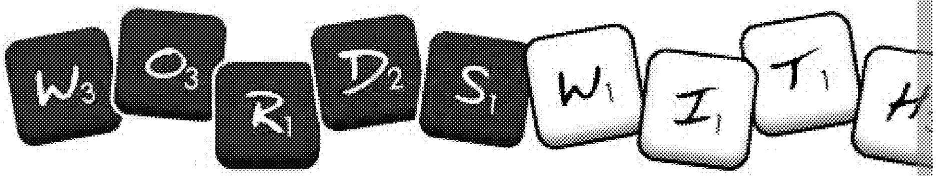
# **PlayGame** and subroutines called from **PlayGame**



## Structure Diagram 3

# HaveTurn and subroutines called from HaveTurn



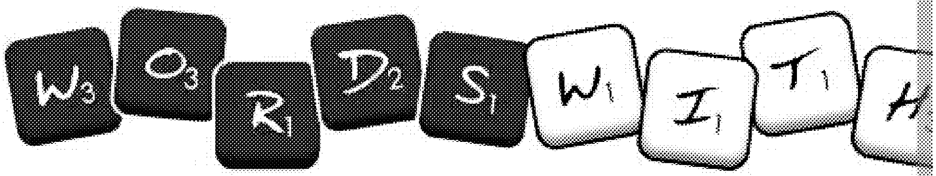


# Written Questions:
# Suggested Answers and Mark Schem

| Q | Answer/Guidance |
|---|---|
| 1a | `QueueOfTiles` |
| 1b | `Contents` |
| 1c | `InTiles/ValidWord` |
| 1d | `TileDictionary` |
| 1e | `CreateTileDictionary/GetChoice/GetNewTileChoice/IsEmpty/` |
| 1f | `DisplayMenu/Add` |
| 1g | `newTileChoice/validChoice/validWord/choice` |
| 1h | `Contents/Rear/MaxSize` |
| 2 | 1 mark each for up to three of the following (NB listed alphabetically).<br>The whole line must be included for the mark.<br>• `AllowedWords.Add(FileReader.ReadLine().Trim().ToUpper())`<br>• `AllowedWords.Clear()`<br>• `Choice = Choice.ToUpper()`<br>• `Console.WriteLine("Points for " & Tile.Key & ": " & Tile.Val`<br>• `Contents(Rear) = Chr(65 + RandNo)`<br>• `CopyOfTiles = Replace(CopyOfTiles, Word(Count), "", , 1)`<br>• `Dim FileReader As New System.IO.StreamReader("aqawords.txt")`<br>• `FileReader.Close()`<br>• `If CopyOfTiles.Contains(Word(Count)) Then`<br>• `PlayerTiles = Replace(PlayerTiles, Letter, "", , 1)`<br>• `RandNo = Int(Rnd() * 26)`<br>• `Randomize()`<br>• `While FileReader.EndOfStream <> True`<br>Also:<br>• Any line containing `Array.IndexOf`<br>• Any line containing `Console.Write / Console.WriteLine / Co`<br>• Any line containing `Len(`<br>• Any line containing `TileDictionary.Add` |
| 3 | 1 mark for each of the:<br>• Adds (an entry) to the dictionary<br>• Entry includes character with ASCII value of 65 + 'count'<br>• Entry includes integer value '1' |

| Q | Answer/Guidance |
|---|---|
| 4 | 1 mark for stating data structure:<br>• Dictionary<br>Up to 2 marks for description:<br>• Contains unique keys<br>• Keys are letters of the alphabet<br>• Each key links to another value<br>• Other value is an integer / the score for that letter |
| 5 | Up to 2 marks for explanation:<br>• Creates a copy of the player's tiles<br>• Characters are eliminated from the copy<br>• Prevents tiles being eliminated from player's tiles (since word has not b |
| 6 | 2 marks for difference between procedure and function:<br>• A *procedure* performs a sequence of events but *does not* return a value<br>• A *function* also performs a sequence of events but *does* return a value<br>Sufficient for 2 marks: a function returns a value – a procedure does not<br>1 mark for identifying a procedure:<br>• `Add / QueueOfTiles.Add`<br>• `AddEndOfTurnTiles`<br>• `DisplayMenu`<br>• `DisplayTilesInHand`<br>• `DisplayTileValues`<br>• `DisplayWinner`<br>• `FillHandWithTiles`<br>• `HaveTurn`<br>• `LoadAllowedWords`<br>• `Main`<br>• `New / QueueOfTiles.New`<br>• `PlayGame`<br>• `Show / QueueOfTiles.Show`<br>• `UpdateAfterAllowedWord`<br>• `UpdateScoreWithPenalty`<br>1 mark for identifying a function:<br>• `CreateTileDictionary`<br>• `GetStartingHand`<br>• `GetChoice`<br>• `CheckWordIsInTiles`<br>• `CheckWordIsValid`<br>• `GetNewTileChoice`<br>• `GetScoreForWord`<br>• `IsEmpty / QueueOfTiles.IsEmpty`<br>• `Remove / QueueOfTilesRemove` |
| 7 | 3 marks:<br>• An exception would occur<br>• Execution would pass to the 'catch' block<br>• The list (AllowedWords) would be cleared |
| 8 | 4 marks:<br>• `AllowedWords` is appended to / new item added to `AllowedWords`<br>• Line is read from a file<br>• White space is removed<br>• Set to upper case / capitalised |

| Q | Answer/Guidance |
|---|---|
| 9 | 2 marks:<br>• A constructor is a method called by the command `new`<br>• Creates a new object based on the class in which it resides |
| 10 | 2 marks:<br>• Sets the instance variable (`MaxSize`)<br>• ... to the parameter (`MaxSize`) |
| 11 | 2 marks:<br>• `Rear` points to the back of the queue<br>• Value of -1 indicates an empty queue<br>• -1 is a rogue value |
| 12 | 1 mark for parameters:<br>• Player's current score, tiles in the player's hand, `Dictionary` that lin<br>Up to 4 marks from the following:<br>• Purpose of function is to subtract value of player's tiles/hand from the<br>• Loop is established to iterate through each tile/character in the player<br>• Value of each tile is determined by looking up in `TileDictionary`<br>• Value is subtracted from player's score<br>• There is no return value...<br>• ...because `PlayerScore` is passed by reference |
| 13 | 4 marks:<br>• Score/integer/variable is set to zero<br>• Loop iterates through each character in the word<br>• Value of character looked up in `TileDictionary` / the `Dictionar`<br>• Value added to score |
| 14 | 2 marks<br>• Loop continues until user has selected '1', '2', '3' or '4'<br>• Validates user input |
| 15 | 3 marks:<br>• Contents of this variable are passed to `AddEndOfTurnTiles`<br>• Selection of '2' indicates that three new tiles will be drawn<br>• The only way to replace this selection is to have played a valid word |
| **TOTAL MARKS** | |



# Programming Tasks:
# Suggested Solutions and Mark Schem

*That the following are recommended solutions, and not an exhaustive list of all possible so*
*guidance should be used as a guide only. Discretion should be used in awarding credit wh*

## Question 1

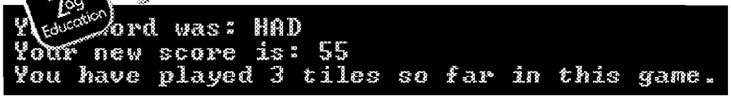
**1 mark** An IF statement that evaluates to TRUE for a word length of either two

**1 mark** The IF statement evaluates to TRUE for a word length of two or three le

**1 mark** Score decremented correctly within the IF statement

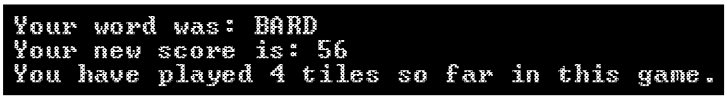
```
If Len(Word) > 7 Then
    Score += 20
ElseIf Len(Word) > 5 Then
    Score += 5
ElseIf Len(Word) = 3 Or Len(Word) = 2 Then
    Score -= 1
End If
```

**1 mark** Screenshot shows 'HAD' was played, the new total is '55' and the total t



```
Your word was: HAD
Your new score is: 55
You have played 3 tiles so far in this game.
```

**1 mark** Screenshot shows 'BARD' was played, the new total is '56' and the total



```
Your word was: BARD
Your new score is: 56
You have played 4 tiles so far in this game.
```

## Question 2

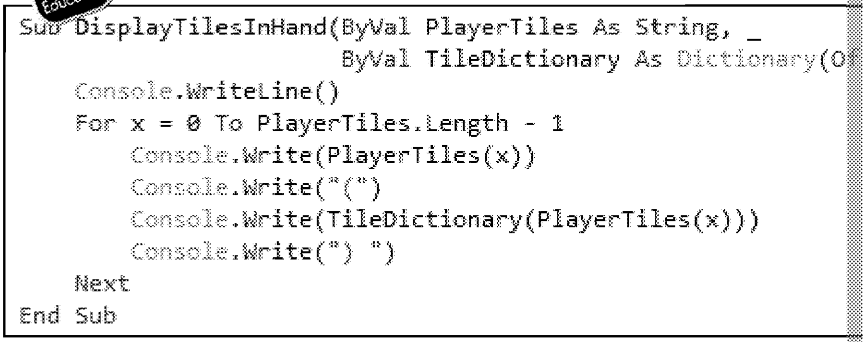
**1 mark** DisplayTilesInHand uses additional parameter of data type Dicti has been used to grant access to the dictionary, but marks 6 and 7 for t available)

**1 mark** Loop to iterate through each character in the player's hand

**1 mark** Display the character (**R**: if the whole hand is still *also* displayed in its o

**1 mark** Display the value of the letter taken from the Dictionary

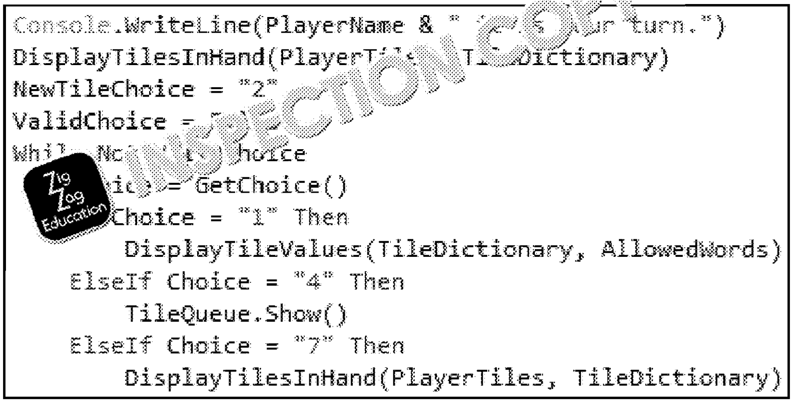
**1 mark** Correct formatting, to include brackets and a single space after each cl



```
Sub DisplayTilesInHand(ByVal PlayerTiles As String, _
                       ByVal TileDictionary As Dictionary(Of
    Console.WriteLine()
    For x = 0 To PlayerTiles.Length - 1
        Console.Write(PlayerTiles(x))
        Console.Write("(")
        Console.Write(TileDictionary(PlayerTiles(x)))
        Console.Write(") ")
    Next
End Sub
```

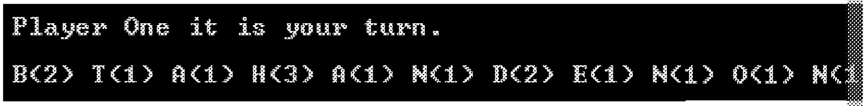
**1 mark** Initial call for player's hand from HaveTurn uses new argument correc

**1 mark** Second call for player's hand from HaveTurn uses new argument corre



```
Console.WriteLine(PlayerName & " it is your turn.")
DisplayTilesInHand(PlayerTiles, TileDictionary)
NewTileChoice = "2"
ValidChoice = False
While Not ValidChoice
    Choice = GetChoice()
    If Choice = "1" Then
        DisplayTileValues(TileDictionary, AllowedWords)
    ElseIf Choice = "4" Then
        TileQueue.Show()
    ElseIf Choice = "7" Then
        DisplayTilesInHand(PlayerTiles, TileDictionary)
```

**1 mark** Screenshot showing correct output format, which should no longer incl



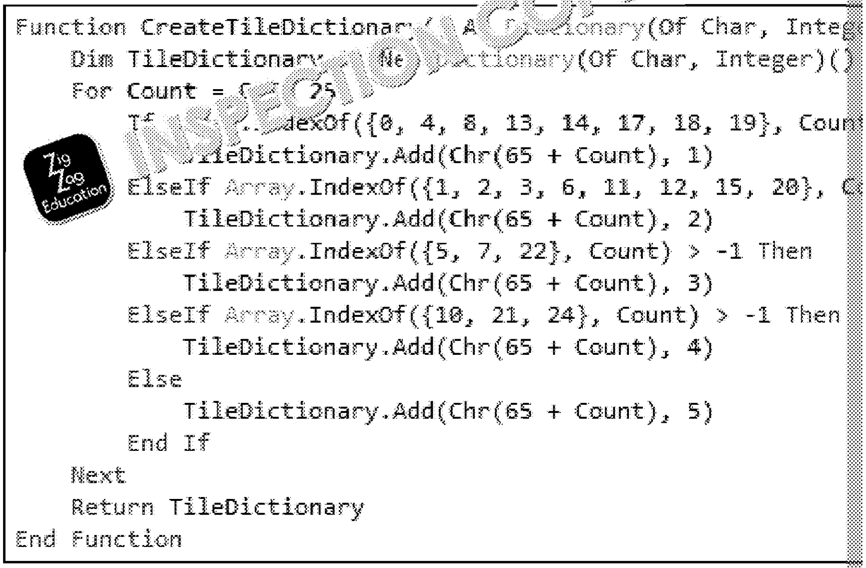
```
Player One it is your turn.
B(2) T(1) A(1) H(3) A(1) N(1) D(2) E(1) N(1) O(1) N(1
```

## Question 3

**1 mark** Removal of values 10, 21 and 24 from the IF structure that assigns thre

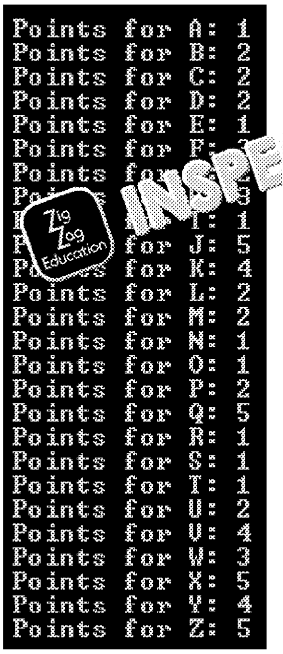
**1 mark** Inclusion of values 10, 21 and 24 in a new selection structure

**1 mark** These values, and only these values, assigned a score of 4



```
Function CreateTileDictionary() As Dictionary(Of Char, Integ
    Dim TileDictionary As New Dictionary(Of Char, Integer)()
    For Count = 0 To 25
        If Array.IndexOf({0, 4, 8, 13, 14, 17, 18, 19}, Coun
            TileDictionary.Add(Chr(65 + Count), 1)
        ElseIf Array.IndexOf({1, 2, 3, 6, 11, 12, 15, 20}, C
            TileDictionary.Add(Chr(65 + Count), 2)
        ElseIf Array.IndexOf({5, 7, 22}, Count) > -1 Then
            TileDictionary.Add(Chr(65 + Count), 3)
        ElseIf Array.IndexOf({10, 21, 24}, Count) > -1 Then
            TileDictionary.Add(Chr(65 + Count), 4)
        Else
            TileDictionary.Add(Chr(65 + Count), 5)
        End If
    Next
    Return TileDictionary
End Function
```

**1 mark** Screenshot shows that K, V and Y are worth four points each



```
Points for A: 1
Points for B: 2
Points for C: 2
Points for D: 2
Points for E: 1
Points for F: 3
Points for G: 2
Points for H: 3
Points for I: 1
Points for J: 5
Points for K: 4
Points for L: 2
Points for M: 2
Points for N: 1
Points for O: 1
Points for P: 2
Points for Q: 5
Points for R: 1
Points for S: 1
Points for T: 1
Points for U: 2
Points for V: 4
Points for W: 3
Points for X: 5
Points for Y: 4
Points for Z: 5
```

## Question 4

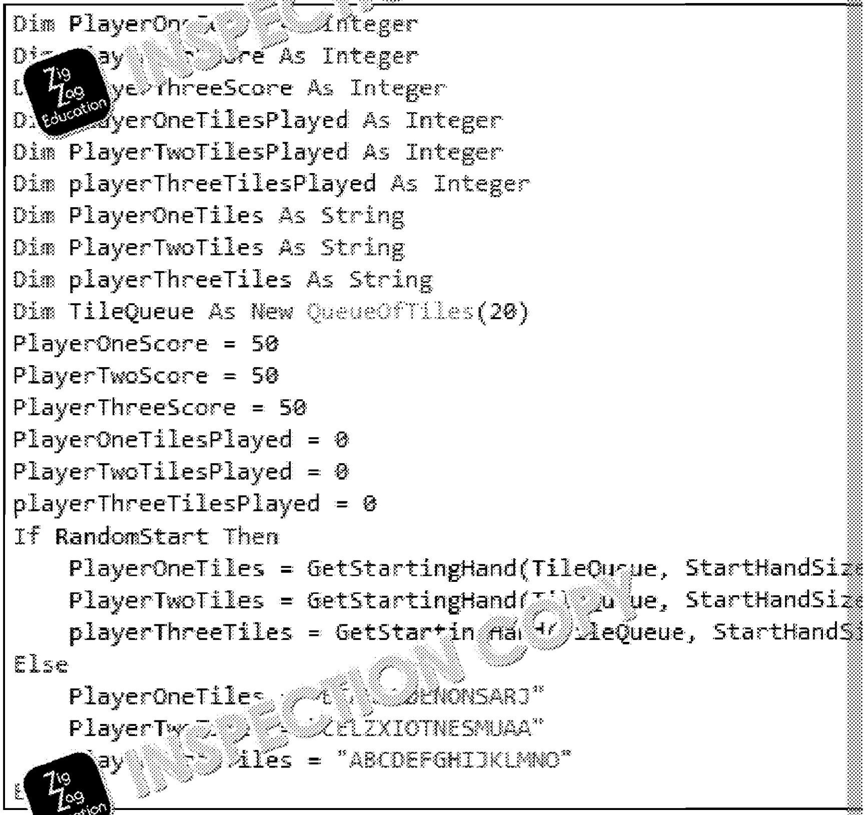
**1 mark** Adding a score for *Player Three* and setting it to 50

**1 mark** Adding a tile count for *Player Three* and setting it to zero

**1 mark** Declaring a variable to hold tiles for *Player Three*

**1 mark** *Player Three*'s hand filled with random tiles in the `If` block

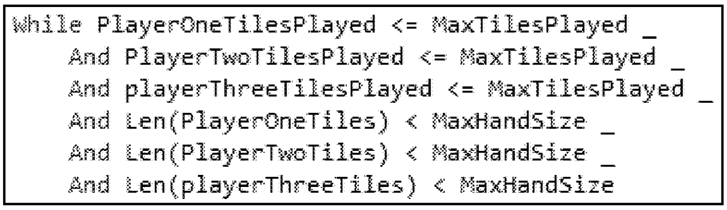
**1 mark** *Player Three*'s hand filled with the letters A–O in the `Else` block



```
Dim PlayerOneScore As Integer
Dim PlayerTwoScore As Integer
Dim PlayerThreeScore As Integer
Dim PlayerOneTilesPlayed As Integer
Dim PlayerTwoTilesPlayed As Integer
Dim playerThreeTilesPlayed As Integer
Dim PlayerOneTiles As String
Dim PlayerTwoTiles As String
Dim playerThreeTiles As String
Dim TileQueue As New QueueOfTiles(20)
PlayerOneScore = 50
PlayerTwoScore = 50
PlayerThreeScore = 50
PlayerOneTilesPlayed = 0
PlayerTwoTilesPlayed = 0
playerThreeTilesPlayed = 0
If RandomStart Then
    PlayerOneTiles = GetStartingHand(TileQueue, StartHandSiz
    PlayerTwoTiles = GetStartingHand(TileQueue, StartHandSiz
    playerThreeTiles = GetStartingHand(TileQueue, StartHandSi
Else
    PlayerOneTiles = "BTAHANDENONSARJ"
    PlayerTwoTiles = "CELZXIOTNESMUAA"
    playerThreeTiles = "ABCDEFGHIJKLMNO"
End If
```

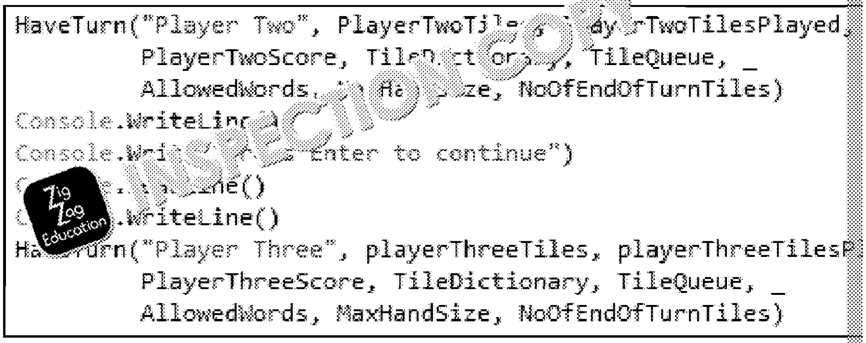
**1 mark** Logic expression in 'while' loop includes tiles played for *Player Three*

**1 mark** Logic expression includes check for size of hand and all logic is sound



```
While PlayerOneTilesPlayed <= MaxTilesPlayed _
    And PlayerTwoTilesPlayed <= MaxTilesPlayed _
    And playerThreeTilesPlayed <= MaxTilesPlayed _
    And Len(PlayerOneTiles) < MaxHandSize _
    And Len(PlayerTwoTiles) < MaxHandSize _
    And Len(playerThreeTiles) < MaxHandSize
```

**1 mark** Call to `HaveTurn` with variables for tiles, tiles played and score for *Pla*



```
HaveTurn("Player Two", PlayerTwoTiles, PlayerTwoTilesPlayed,
         PlayerTwoScore, TileDictionary, TileQueue, _
         AllowedWords, MaxHandSize, NoOfEndOfTurnTiles)
Console.WriteLine()
Console.WriteLine("Press Enter to continue")
Console.ReadLine()
Console.WriteLine()
HaveTurn("Player Three", playerThreeTiles, playerThreeTilesP
         PlayerThreeScore, TileDictionary, TileQueue, _
         AllowedWords, MaxHandSize, NoOfEndOfTurnTiles)
```

**1 mark** Call to `UpdateScoreWithPenalty` for *Player Three*

**1 mark** Call to `DisplayWinner` passes scores for all three players

```
UpdateScoreWithPenalty(PlayerOneScore, PlayerOneTiles, TileD
UpdateScoreWithPenalty(PlayerTwoScore, PlayerTwoTiles, TileD
UpdateScoreWithPenalty(PlayerThreeScore, playerThreeTiles, T
DisplayWinner(PlayerOneScore, PlayerTwoScore, PlayerThreeSco
```

**1 mark** Subroutine `DisplayWinner` requires three parameters instead of two

**1 mark** Score for *Player Three* is displayed

```
Sub DisplayWinner(ByVal PlayerOneScore As Integer, ByVal Pla
                  ByVal PlayerThreeScore As Integer)
    Console.WriteLine()
    Console.WriteLine("**** GAME OVER! ****")
    Console.WriteLine()
    Console.WriteLine("Player One your score is " & PlayerOne
    Console.WriteLine("Player Two your score is " & PlayerTwo
    Console.WriteLine("Player Three your score is " & PlayerT
```

**1 mark** Correct logic for displaying 'Player One wins!'

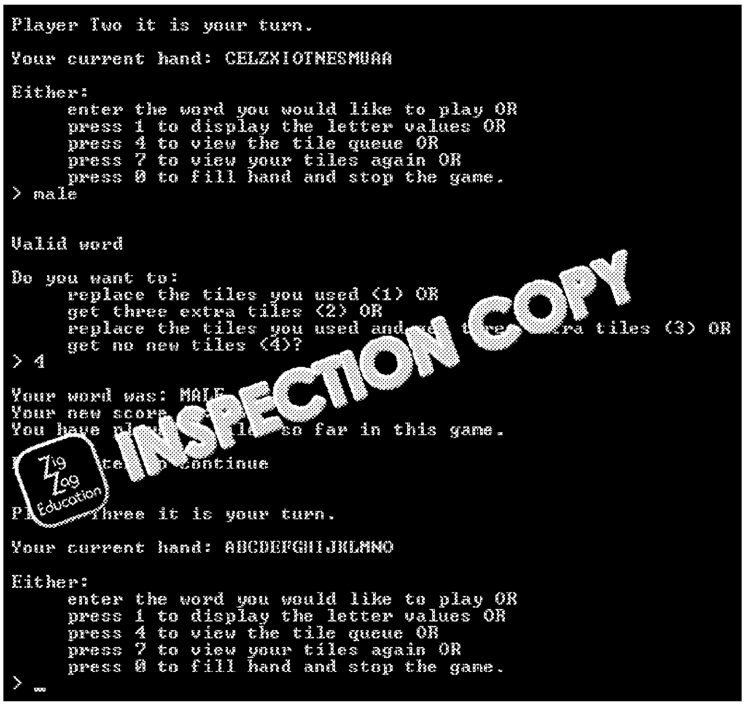
**1 mark** Correct logic for displaying 'Player Two wins!'

**1 mark** Correct logic for displaying 'Player Three wins!'

**1 mark** 'No clear winner' displayed in 'else' block (**A:** if this has been written as covers all other combinations; **R:** if other text is displayed)

```
If PlayerOneScore > PlayerTwoScore And PlayerOneScore > Play
    Console.WriteLine("Player One wins!")
ElseIf PlayerTwoScore > PlayerOneScore And PlayerTwoScore > P
    Console.WriteLine("Player Two wins!")
ElseIf PlayerThreeScore > PlayerOneScore And PlayerThreeScor
    Console.WriteLine("Player Three wins!")
Else
    Console.WriteLine("No clear winner")
End If
```

**1 mark** Prompt for *Player Three* to move after *Player Two* has moved with *Player* (ABCDEFGHIJKLMNO)



```
Player Two it is your turn.

Your current hand: CELZXIOTNESMUAA

Either:
     enter the word you would like to play OR
     press 1 to display the letter values OR
     press 4 to view the tile queue OR
     press 7 to view your tiles again OR
     press 0 to fill hand and stop the game.
> male


Valid word

Do you want to:
     replace the tiles you used (1) OR
     get three extra tiles (2) OR
     replace the tiles you used and get three extra tiles (3) OR
     get no new tiles (4)?
> 4

Your word was: MALE
Your new score [illegible]
You have played [illegible] tiles so far in this game.

Press Enter to continue


Player Three it is your turn.

Your current hand: ABCDEFGHIJKLMNO

Either:
     enter the word you would like to play OR
     press 1 to display the letter values OR
     press 4 to view the tile queue OR
     press 7 to view your tiles again OR
     press 0 to fill hand and stop the game.
> _
```

## Question 5

**1 mark** Use of a variable to store the score for the word (**A:** if no variable is use GetScoreForWord forms part of the string concatenation to display ' 30 points.')

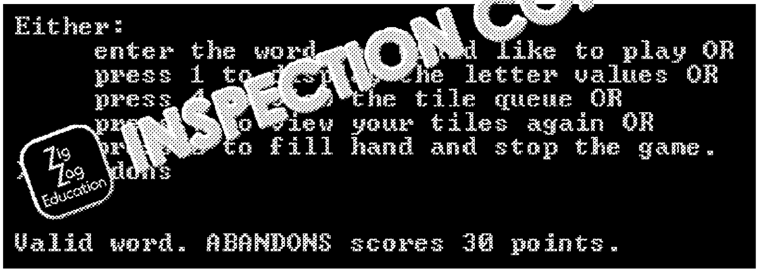
**1 mark** Call to GetScoreForWord to either initialise the variable or place the concatenated string

**1 mark** Correct parameters – Choice, TileDictionary

**1 mark** Concatenation incorporates all components stated in the question, inclu (A: full stop is not included, difference in case). Concatenation mus the variable used to store the score (if a variable was used).

```
If ValidWord Then

    Dim currentScore As Integer = GetScoreForWord(Choice, Ti

    Console.WriteLine()
    Console.WriteLine("Valid word. " & Choice & " scores " &
    Console.WriteLine()
    UpdateAfterAllowedWord(Choice, PlayerTiles, PlayerScore,
                           TileDictionary, AllowedWords)
    NewTileChoice = GetNewTileChoice()
End If
```

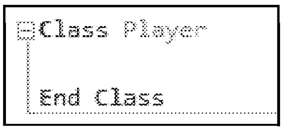
**1 mark** Input of word 'abandons' displays score worth 30 points (**DPT:** spacing e



```
Either:
     enter the word you would like to play OR
     press 1 to display the letter values OR
     press 4 to view the tile queue OR
     press 7 to view your tiles again OR
     press 0 to fill hand and stop the game.
> abandons

Valid word. ABANDONS scores 30 points.
```

## Question 6

**1 mark** Class declaration (**R:** if name or case incorrect)

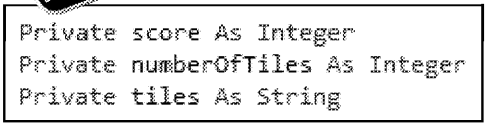


```
Class Player

End Class
```

**1 mark** Attributes declared with appropriate names (**A:** alternative names if mea

**1 mark** All three attributes have correct data types (**R:** if any additional attributes

**1 mark** Attributes declared `private`



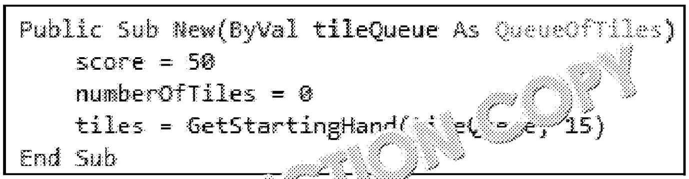
```
Private score As Integer
Private numberOfTiles As Integer
Private tiles As String
```

**1 mark** Constructor declared with single parameter `QueueOfTiles tileQue`

**1 mark** `score` and `numberOfTiles` (or their equivalents) initialised to `50` an

**1 mark** `tiles` (or its equivalent) initialised using a call to `GetStartingHan`

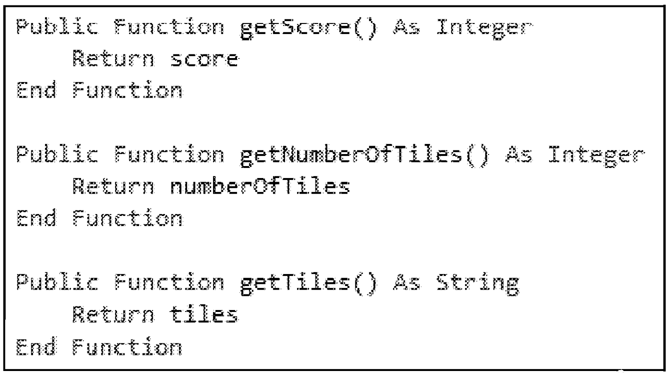
**1 mark** `GetStartingHand` contains correct parameters (**A:** integers other tha



```
Public Sub New(ByVal tileQueue As QueueOfTiles)
    score = 50
    numberOfTiles = 0
    tiles = GetStartingHand(tileQueue, 15)
End Sub
```

**1 mark** Functions with appropriate data types to return all attributes (**DPT:** extr

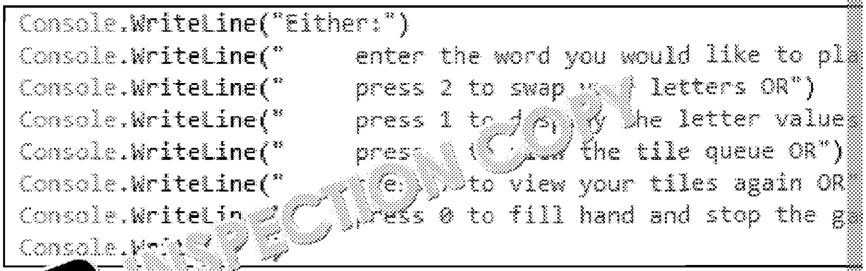
**1 mark** Functions all declared `public`



```
Public Function getScore() As Integer
    Return score
End Function

Public Function getNumberOfTiles() As Integer
    Return numberOfTiles
End Function

Public Function getTiles() As String
    Return tiles
End Function
```

## Question 7

**1 mark** Addition of the extra option in `GetChoice`



```
Console.WriteLine("Either:")
Console.WriteLine("     enter the word you would like to pla
Console.WriteLine("     press 2 to swap your letters OR")
Console.WriteLine("     press 1 to display the letter values
Console.WriteLine("     press 4 to view the tile queue OR")
Console.WriteLine("     press 7 to view your tiles again OR"
Console.WriteLine("     press 0 to fill hand and stop the ga
Console.Wri
```

**1 mark** Inclusion of 'else if' clause to deal with 'choice' being '2'

**1 mark** Variable to temporarily store the new letters (**A:** valid repurposing of pl

**1 mark** Loop that runs once per letter in the original hand

**1 mark** Calling `remove` and appending the character to the player's hand

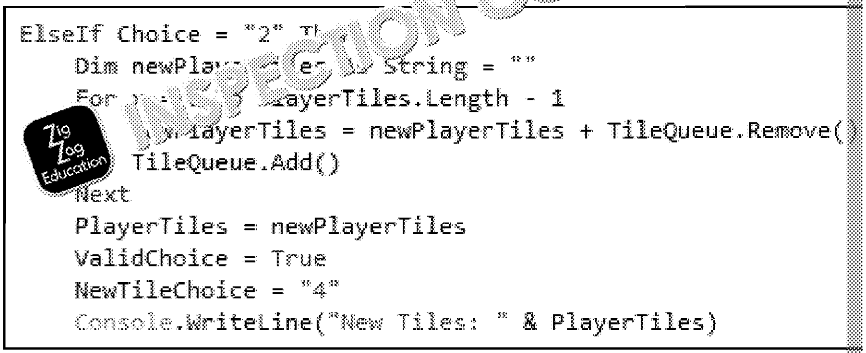
**1 mark** Calling `add` to keep the tile queue full

**1 mark** `playerTiles` contains the new letters

**1 mark** `validChoice` set to 'true' to prevent the main loop in `HaveTurn` repe

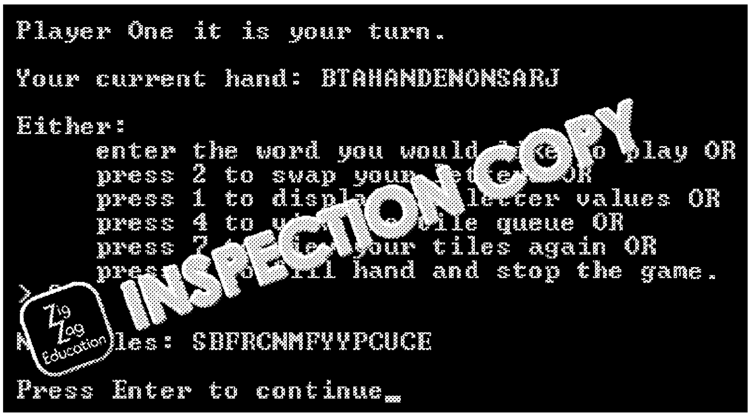
**1 mark** `newTileChoice` set to '4' to ensure that no new tiles are taken on top

**1 mark** Output of new hand



```
ElseIf Choice = "2" Then
    Dim newPlayerTiles As String = ""
    For x = 0 To PlayerTiles.Length - 1
        newPlayerTiles = newPlayerTiles + TileQueue.Remove(
        TileQueue.Add()
    Next
    PlayerTiles = newPlayerTiles
    ValidChoice = True
    NewTileChoice = "4"
    Console.WriteLine("New Tiles: " & PlayerTiles)
```

**1 mark** Output to show the original hand, selection of '2' from the menu and th comprise random letters)



```
Player One it is your turn.

Your current hand: BTAHANDENONSARJ

Either:
     enter the word you would like to play OR
     press 2 to swap your letters OR
     press 1 to display the letter values OR
     press 4 to view the tile queue OR
     press 7 to view your tiles again OR
     press 0 to fill hand and stop the game.
> 2

New Tiles: SBFRCNMFYYPCUCE

Press Enter to continue_
```

## Question 8

**1 mark** Character attribute in `QueueOfTiles` of suitable name (**I:** access modi

```
Protected Contents() As Char
Protected Rear As Integer
Protected MaxSize As Integer
Protected lastLetterAdded As Char
```

**1 mark** Loop inside existing 'if' structure to iterate until a duplicate has not bee intent of loop is clear

**1 mark** Generation of a new letter exists inside the loop

**1 mark** Comparison of new letter and previous letter is made inside the loop

**1 mark** Incrementation of 'rear' outside the loop

**1 mark** Termination of loop depends on correct comparison of new letter and t

**1 mark** Attribute is set to the most recent new letter by the end of the subrouti

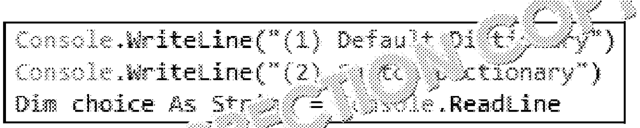
```
Public Sub Add()
    Dim RandNo As Integer
    If Rear < MaxSize - 1 Then
        Rear += 1
        Dim duplicate As Boolean = True
        While duplicate
            RandNo = Int(Rnd() * 26)
            Contents(Rear) = Chr(65 + RandNo)
            If Not (Contents(Rear) = lastLetterAdded) Then
                duplicate = False
            End If
        End While
        lastLetterAdded = Contents(Rear)
    End If
End Sub
```

## Question 9

**1 mark** Options displayed to user

**1 mark** Declaration of variable to store input from this menu

**1 mark** Input assigned to variable



```
Console.WriteLine("(1) Default Dictionary")
Console.WriteLine("(2) Custom Dictionary")
Dim choice As String = Console.ReadLine
```

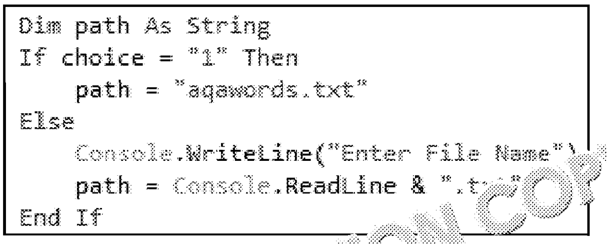
**1 mark** Declaration of variable to store file path (**A:** input concatenated directly within the try block)

**1 mark** A choice of '1' on the previous menu will cause `aqawords.txt` to be l

**1 mark** A choice of '2' results in the user being prompted for a path

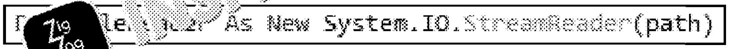
**1 mark** User response is placed into the appropriate variable or concatenated d within the try block

**1 mark** String '.txt' appended to the user input file



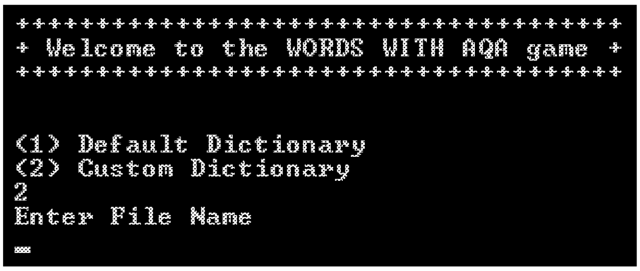
```
Dim path As String
If choice = "1" Then
    path = "aqawords.txt"
Else
    Console.WriteLine("Enter File Name")
    path = Console.ReadLine & ".txt"
End If
```

**1 mark** File path specified by the user is accessed



```
Dim fileReader As New System.IO.StreamReader(path)
```

**1 mark** Output shows the new menu, selection of '2' and prompting for the file



```
++++++++++++++++++++++++++++++++++++++++
+ Welcome to the WORDS WITH AQA game +
++++++++++++++++++++++++++++++++++++++++


(1) Default Dictionary
(2) Custom Dictionary
2
Enter File Name
_
```

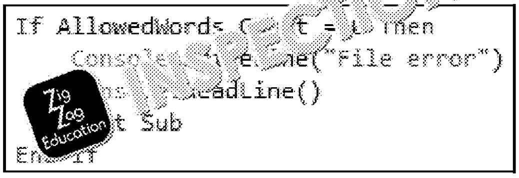
## Question 10

**1 mark** 'if' clause after call to `LoadAllowedWords`

**1 mark** 'if' clause compares array length correctly, i.e.= 0 or <1

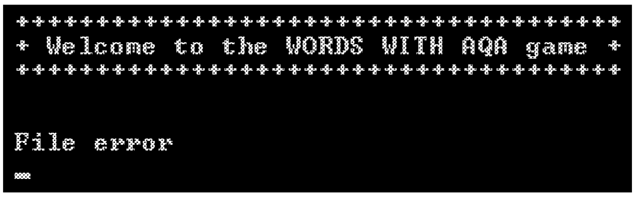
**1 mark** Correct output message

**1 mark** Instruction to end program or subroutine



```
If AllowedWords.Count = 0 Then
   Console.WriteLine("File error")
   Console.ReadLine()
   Exit Sub
End If
```

**1 mark** Output displays welcome message followed by 'file error'



```
*****************************************
* Welcome to the WORDS WITH AQA game *
*****************************************


File error
_
```

## Question 11

**1 mark** Message correctly placed (even if message is incorrect) before call to Ge

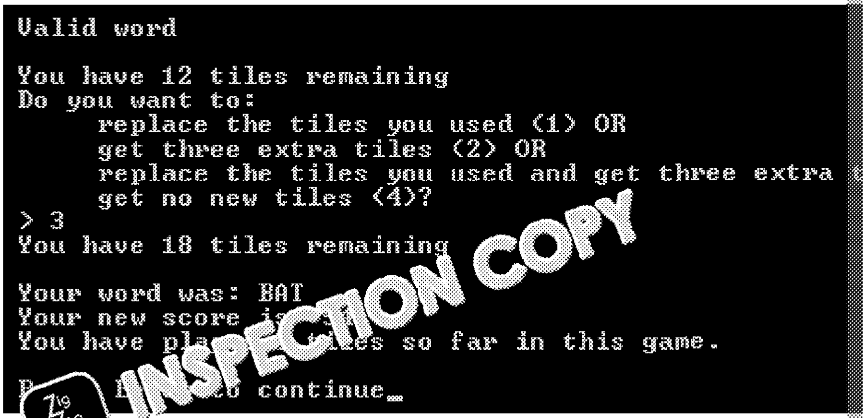
```
UpdateAfterAllowedWord(Choice, PlayerTiles, PlayerScore, _
                       PlayerTilesPlayed, TileDictionary, All
Console.WriteLine("You have " & PlayerTiles.Length & " tiles
NewTileChoice = GetNewTileChoice()
```

**1 mark** Message correctly placed (even if message is incorrect) immediately aft AddEndOfTurnTiles (A... code is added outside the 'if' clause, but o clause to prevent the message being redisplayed in the event that no n

**1 mark** String concatenation to display message in the specified format in both

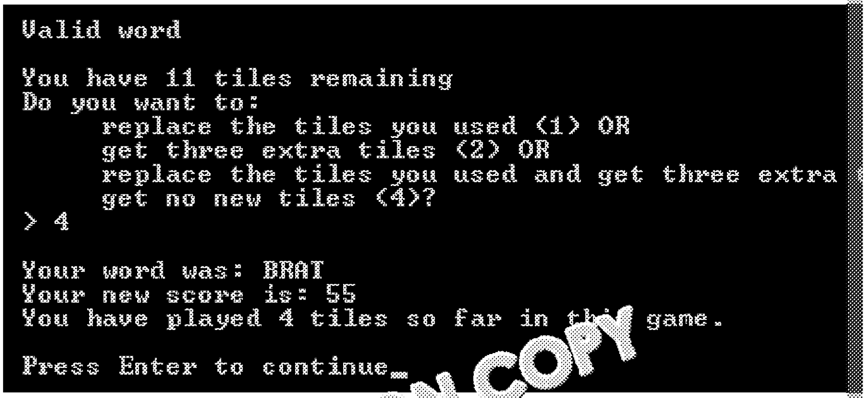
```
If NewTileChoice <> "4" Then
    AddEndOfTurnTiles(TileQueue, PlayerTiles, NewTileChoice,
    Console.WriteLine("You have " & PlayerTiles.Length & " t
End If
```

**1 mark** Output showing 12 tiles before the draw and 18 afterwards



```
Valid word

You have 12 tiles remaining
Do you want to:
     replace the tiles you used (1) OR
     get three extra tiles (2) OR
     replace the tiles you used and get three extra t
     get no new tiles (4)?
> 3
You have 18 tiles remaining

Your word was: BAT
Your new score is [illegible]
You have played [illegible] tiles so far in this game.

Press Enter to continue_
```

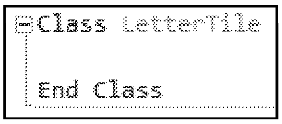
**1 mark** Output showing 11 tiles before the draw, then not displaying the messa be 'Your word was: BRAT'



```
Valid word

You have 11 tiles remaining
Do you want to:
     replace the tiles you used (1) OR
     get three extra tiles (2) OR
     replace the tiles you used and get three extra t
     get no new tiles (4)?
> 4

Your word was: BRAT
Your new score is: 55
You have played 4 tiles so far in this game.

Press Enter to continue_
```

## Question 12

**1 mark** Class declaration (**R:** if name or case incorrect)

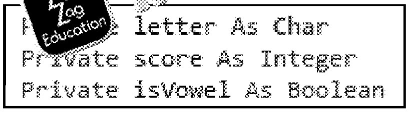


```
Class LetterTile

End Class
```

**1 mark** Attributes declared with appropriate names (**R:** if any other names are u

**1 mark** All attributes use correct data types

**1 mark** Attributes declared private



```
Private letter As Char
Private score As Integer
Private isVowel As Boolean
```

**1 mark** Constructor written as `New`

**1 mark** char parameter (**A:** if named other than 'letter')

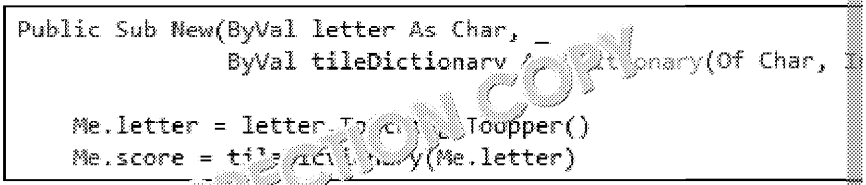
**1 mark** Dictionary parameter (**A:** if named other than '`tileDictionary`')

**1 mark** `letter` attribute set using char parameter

**1 mark** Attribute always contains upper-case version of character

**1 mark** `score` attribute set by extracting a value from the dictionary (even if val

**1 mark** `score` would always be correct, so the parameter passed to the dictionary

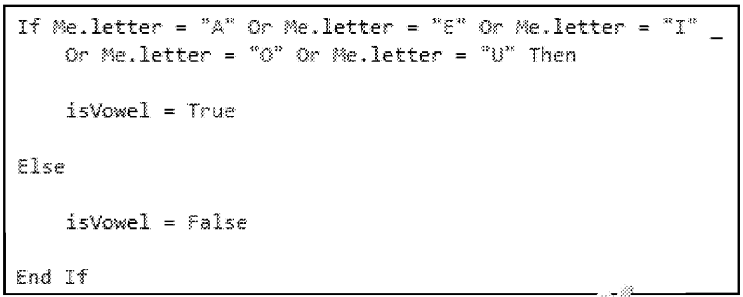


```
Public Sub New(ByVal letter As Char, _
               ByVal tileDictionary As Dictionary(Of Char, I

    Me.letter = letter.ToString.Toupper()
    Me.score = tileDictionary(Me.letter)
```

**1 mark** Selection to isolate either vowels or consonants (even if it would not w

**1 mark** Selection would always isolate vowels/consonants, bearing in mind tha upper case or lower case

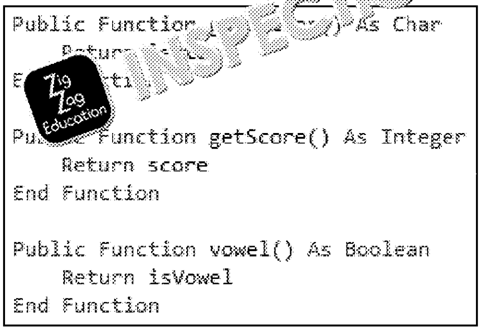
**1 mark** `isVowel` attribute set correctly with this selection



```
If Me.letter = "A" Or Me.letter = "E" Or Me.letter = "I" _
    Or Me.letter = "O" Or Me.letter = "U" Then

    isVowel = True

Else

    isVowel = False

End If
```

**1 mark** Functions with appropriate data types (**A:** any sensible names)

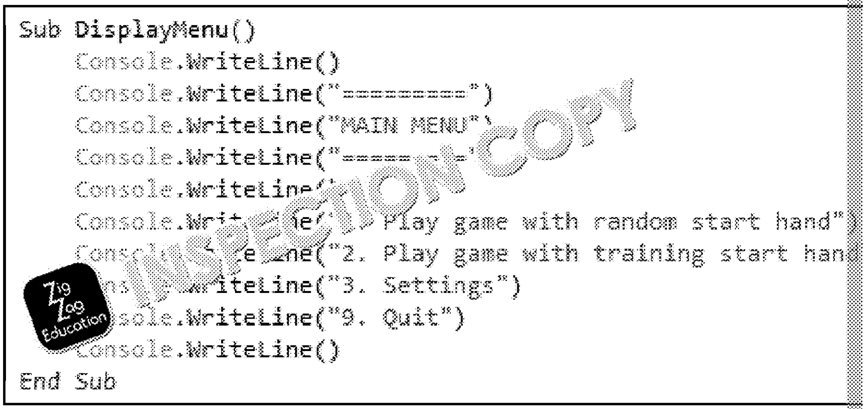
**1 mark** Functions declared public



```
Public Function getLetter() As Char
    Return letter
End Function

Public Function getScore() As Integer
    Return score
End Function

Public Function vowel() As Boolean
    Return isVowel
End Function
```

## Question 13

**1 mark** New entry in `DisplayMenu`

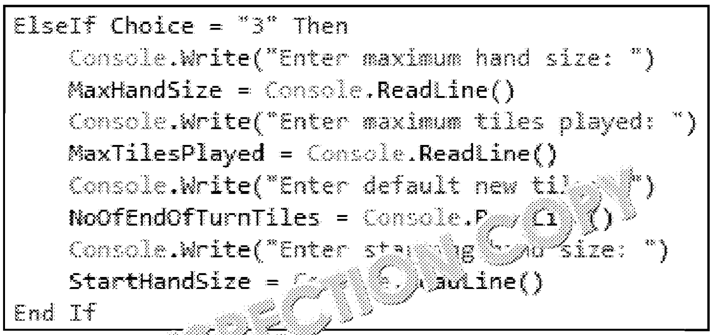


```
Sub DisplayMenu()
    Console.WriteLine()
    Console.WriteLine("=========")
    Console.WriteLine("MAIN MENU")
    Console.WriteLine("=========")
    Console.WriteLine()
    Console.WriteLine("1. Play game with random start hand")
    Console.WriteLine("2. Play game with training start hand
    Console.WriteLine("3. Settings")
    Console.WriteLine("9. Quit")
    Console.WriteLine()
End Sub
```

**1 mark** 'else if' added to main to detect entry of '3'

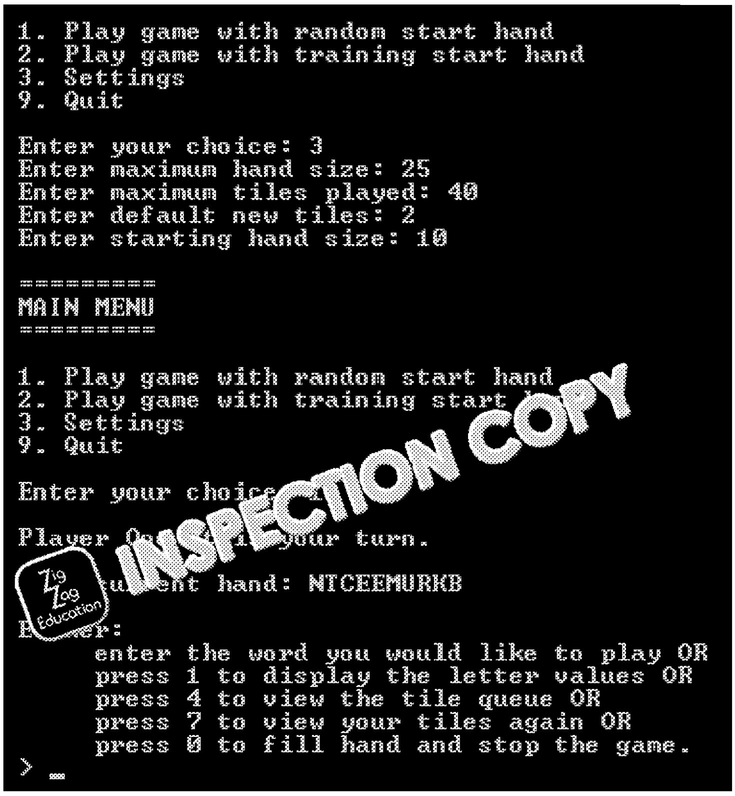
**1 mark** Prompts for all four new inputs (**R:** alternative wording)

**1 mark** Input prompts relate correctly to all four variables



```
ElseIf Choice = "3" Then
    Console.Write("Enter maximum hand size: ")
    MaxHandSize = Console.ReadLine()
    Console.Write("Enter maximum tiles played: ")
    MaxTilesPlayed = Console.ReadLine()
    Console.Write("Enter default new tiles: ")
    NoOfEndOfTurnTiles = Console.ReadLine()
    Console.Write("Enter starting hand size: ")
    StartHandSize = Console.ReadLine()
End If
```

**1 mark** Output showing values 25, 40, 2 and 10 entered, with 10 indicating the hand should now be 10 characters long instead of 15



```
1. Play game with random start hand
2. Play game with training start hand
3. Settings
9. Quit

Enter your choice: 3
Enter maximum hand size: 25
Enter maximum tiles played: 40
Enter default new tiles: 2
Enter starting hand size: 10

=========
MAIN MENU
=========

1. Play game with random start hand
2. Play game with training start hand
3. Settings
9. Quit

Enter your choice: 1

Player One it is your turn.

Current hand: NTCEEMURKB

Enter:
     enter the word you would like to play OR
     press 1 to display the letter values OR
     press 4 to view the tile queue OR
     press 7 to view your tiles again OR
     press 0 to fill hand and stop the game.
> _
```

## Question 14

**1 mark** Variable to track the iterations so that a vowel is guaranteed every third

```
Protected Contents() As Char
Protected Rear As Integer
Protected MaxSize As Integer
Protected vowelCounter As Integer = 0
```

**1 mark** `Rear` is still incremented before any letter is added

**1 mark** Selection structure to determine whether to add a vowel or a random le every third l

**1 mark** New variable is changed to ensure the switch between selecting a vowe (`vowelCounter+=1` in this example, but any equivalent approach can

**1 mark** Random letter still correctly added to the array

```
Public Sub Add()
    Dim RandNo As Integer
    If Rear < MaxSize - 1 Then
        Rear += 1
        If vowelCounter < 2 Then
            RandNo = Int(Rnd() * 26)
            Contents(Rear) = Chr(65 + RandNo)
            vowelCounter += 1
```

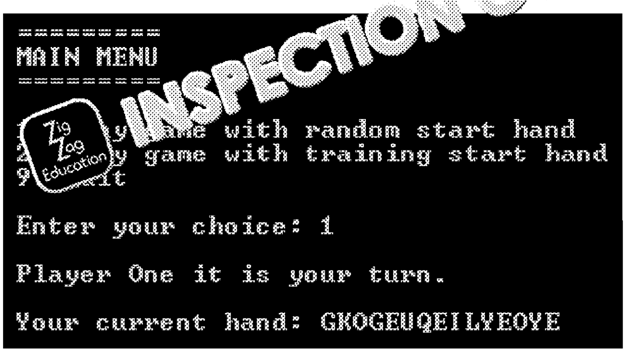
**1 mark** Random number generator selects from vowels, giving each equal prob

**1 mark** Vowel is correctly added to the array

**1 mark** Variable is changed to ensure that the next selection will be a random l

```
Else
    RandNo = Int(Rnd() * 5)
    Select Case RandNo
        Case Is = 0
            Contents(Rear) = Chr(65)
        Case Is = 1
            Contents(Rear) = Chr(69)
        Case Is = 2
            Contents(Rear) = Chr(73)
        Case Is = 3
            Contents(Rear) = Chr(79)
        Case Else
            Contents(Rear) = Chr(85)
    End Select
    vowelCounter = 0
End If
```

**1 mark** Selecting the random starting hand should display every third letter as a

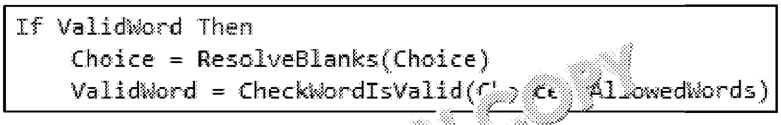


```
=========
MAIN MENU
=========

1. Play game with random start hand
2. Play game with training start hand
9. Quit

Enter your choice: 1

Player One it is your turn.

Your current hand: GKOGEUQEILYEOYE
```

## Question 15a

**1 mark** Call to `ResolveBlanks`, with choice as parameter, before call to `Che`

**1 mark** Value returned from call to `ResolveBlanks` stored in choice



```
If ValidWord Then
    Choice = ResolveBlanks(Choice)
    ValidWord = CheckWordIsValid(Choice, AllowedWords)
```

**1 mark** Method declaration for `ResolveBlanks` with a string return type (**R:** a variation in case')

**1 mark** String parameter (**I:** if any other parameters)

**1 mark** Variable to store user input for the value of a blank tile

**1 mark** Loop to ensure all dashes are found (**R:** if program would fail in the abs

**1 mark** Selection statement to handle the presence of a dash

**1 mark** User is prompted with 'Enter value of blank tile:' (**R:** alternative wordin

**1 mark** User input is stored in variable

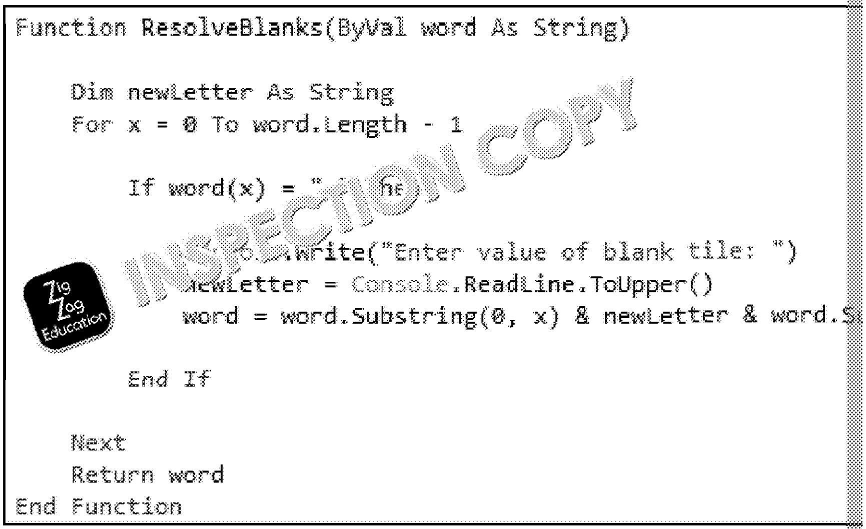
**1 mark** Input is converted to upper case

**1 mark** Attempt to incorporate the new input to replace the dash (even if unsuc

**1 mark** Dash would be replaced by the user input in all cases

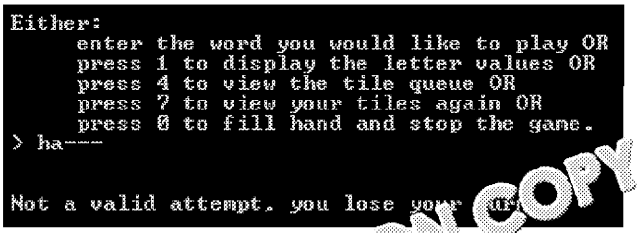
**1 mark** Structure ensures that all dashes would be replaced by characters enter

**1 mark** String correctly returned



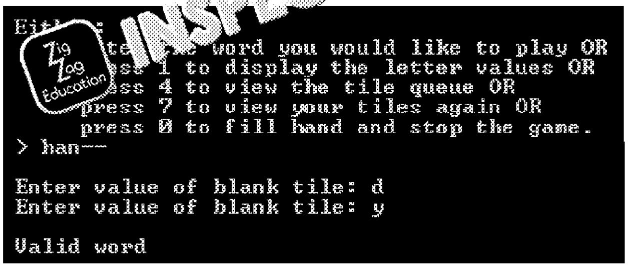
```
Function ResolveBlanks(ByVal word As String)

    Dim newLetter As String
    For x = 0 To word.Length - 1

        If word(x) = "-" Then

            Console.Write("Enter value of blank tile: ")
            newLetter = Console.ReadLine.ToUpper()
            word = word.Substring(0, x) & newLetter & word.S

        End If

    Next
    Return word
End Function
```

**1 mark** Output for `ha---` results in 'Not a valid attempt, you lose your turn.'



```
Either:
     enter the word you would like to play OR
     press 1 to display the letter values OR
     press 4 to view the tile queue OR
     press 7 to view your tiles again OR
     press 0 to fill hand and stop the game.
> ha---

Not a valid attempt, you lose your turn.
```

**1 mark** Output for `han--`, followed by entering 'd' then 'y', results in 'Valid wor



```
Either:
     enter the word you would like to play OR
     press 1 to display the letter values OR
     press 4 to view the tile queue OR
     press 7 to view your tiles again OR
     press 0 to fill hand and stop the game.
> han--

Enter value of blank tile: d
Enter value of blank tile: y

Valid word
```

## Question 15b

**1 mark** Additional parameter in `UpdateAfterAllowedWord` to contain word

**1 mark** This new parameter is iterated through to remove tiles from the player's

```
Sub UpdateAfterAllowedWord(ByVal wordWithBlanks As String, B
                           ByRef PlayerTiles As String, ByRe
                           ByRef PlayerTilesPlayed As Intege
                           ByVal TileDictionary As Dictiona
                           ByVal AllowedWords As List(Of St
    PlayerTilesPlayed += Len(Word)
    For Each Letter In wordWithBlanks
        PlayerTiles = Replace(PlayerTiles, Letter, "", , 1)
    Next
    PlayerScore += GetScoreForWord(Word, TileDictionary)
End Sub
```

**1 mark** Extra variable alongside Choice in `HaveTurn` - one will contain a word the other will contain that word with the blank tiles resolved

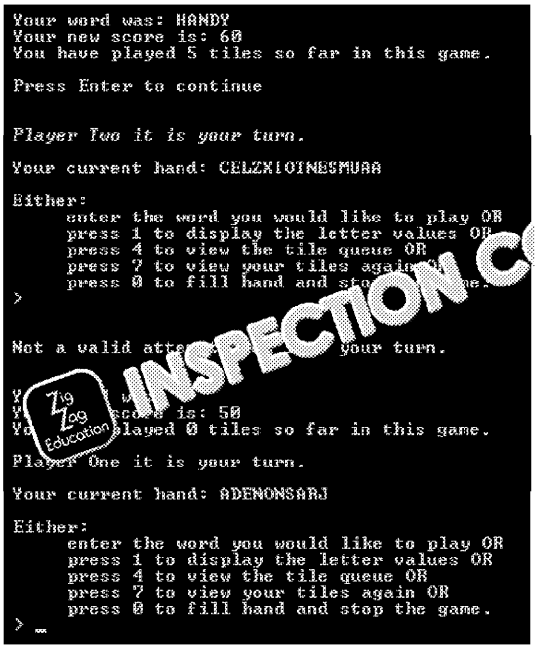
```
While Not ValidChoice
    Choice = GetChoice()
    Dim wordWithBlanks As String = Choice
```

**1 mark** Call to `ResolveBlanks` that results in one of the two variables contai contains the word with the blanks resolved

**1 mark** Call to `UpdateAfterAllowedWord` with the new parameter included

```
If ValidWord Then
    Choice = ResolveBlanks(wordWithBlanks)
    ValidWord = CheckWordIsValid(Choice, AllowedWords)
    If ValidWord Then
        Console.WriteLine()
        Console.WriteLine("Valid word")
        Console.WriteLine()
        UpdateAfterAllowedWord(wordWithBlanks, Choice, Playe
                               PlayerScore, PlayerTilesPlayed
                               TileDictionary, AllowedWords)
        NewTileChoice = GetNewTileChoice()
    End If
End If
```

**1 mark** *Player One*'s hand at start of second turn contains ADENONSARJ



```
Your word was: HANDY
Your new score is: 60
You have played 5 tiles so far in this game.

Press Enter to continue


Player Two it is your turn.

Your current hand: CELZXIOINESMUAA

Either:
      enter the word you would like to play OR
      press 1 to display the letter values OR
      press 4 to view the tile queue OR
      press 7 to view your tiles again OR
      press 0 to fill hand and stop the game.
>


Not a valid attempt, you lose your turn.


Y[illegible] w[illegible]
Your new score is: 50
You have played 0 tiles so far in this game.

Player One it is your turn.

Your current hand: ADENONSARJ

Either:
      enter the word you would like to play OR
      press 1 to display the letter values OR
      press 4 to view the tile queue OR
      press 7 to view your tiles again OR
      press 0 to fill hand and stop the game.
> _
```

| Name | |
|---|---|

ZigZag Education supporting

# A Level AQA Computer Science Pap

## Summer 2018

## Electronic Answer Document (EAD)

**Instructions**

- Enter your name in the box at the top of this page
- Answer **all** questions by entering your answers into this document
- Remember to **save** this document regularly
- Save and print this document and any additional pages

- Answer **all** questions
- The marks available for each question are shown in brackets

- You will need:
  - □ access to a computer
  - □ access to a printer
  - □ access to appropriate software
  - □ electronic copies of the required skeleton code
  - □ EAD (Electronic Answer Document)

**Total marks:**

## Written Questions

Answer all questions.
Remember to save this document regularly.

| Q | Answer | |
|---|---|---|
| 1 | (a) | |
| | (b) | |
| | (c) | |
| | (d) | |
| | (e) | |
| | (f) | |
| | (g) | |
| | (h) | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | |
| 10 | | |
| 11 | | |
| 12 | | |
| 13 | | |
| 14 | | |
| 15 | | |

## Programming Tasks

Answer all questions.
Remember to save this document regularly.

| Q | Answer |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |
| 10 | |
| 11 | |
| 12 | |
| 13 | |
| 14 | |
| 15 | |